João Miguel de Matos Gil Ramos

BSc.

Livefeeds - Desenho e Avaliação de um Algoritmo de Filiação com Visibilidade Parcial

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Professor Doutor Sérgio Duarte, Prof. Auxiliar, Universidade Nova de Lisboa

Júri:

Presidente: Doutor Francisco de Moura e Castro Ascensão de Azevedo

Arguente: Doutor Hugo Alexandre Tavares Miranda

Vogal: Doutor Sérgio Marco Duarte

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2012

Livefeeds - Desenho e Avaliação de um Algoritmo de Filiação com Visibilidade Parcial

Agradecimentos

Ao Professor Sérgio Duarte, meu orientador, pelo apoio e tempo disponibilizado.
Aos meus Pais que sempre me incentivaram e apoiaram.
À Mariana, que faz de mim uma pessoa melhor todos os dias.
Ao meu Avô Orlando, à memória de quem dedico este trabalho, por todo o incentivo, confiança e por ser um exemplo a seguir.

Resumo

A Internet oferece-nos hoje um número infinito de fontes de conteúdos. Perante este vasto universo de informação, é difícil ao utilizador manter-se em contacto com todas as suas fontes de interesse uma vez que este contacto envolve muitas vezes uma procura activa por parte do utilizador.

Sistemas editor/assinante implementam a relação entre o utilizador como consumidor de informação e as fontes de informação, permitindo numa vertente orientada pelo conteúdo que este defina subscrições detalhando os seus interesses. Soluções centralizadas são frequentes mas limitadas em escala e dispendiosas. Mais ainda, soluções centralizadas raramente envolvem a notificação dos utilizadores e dependem de *polling* constante ao servidor.

O Projecto Livefeeds oferece uma alternativa descentralizada baseada numa aproximação par-a-par (*peer-to-peer*) em que os seus participantes cooperam na disseminação de mensagens de acordo com os filtros estipulados previamente pelos seus participantes. Esta disseminação, conhecida como o problema da disseminação filtrada, deve garantir que todos os nós recebem as mensagens nas quais estipularam interesse e que não recebem mensagens indesejadas.

A actual solução baseia-se num algoritmo de filiação que cria uma visão completa do sistema em cada nó e um algoritmo de disseminação que faz uso desta visão na altura das disseminações de eventos. Embora esta visão permita garantir as condições referidas, manter uma visão completa dum sistema com alto dinamismo apresenta um custo elevado.

Em alternativa, um algoritmo de filiação para uma visão parcial pode ser implementado. Em teoria uma redução da visão mantida em cada nó permitirá reduzir o custo associado à entrada de cada nó de forma linear em relação a essa mesma redução.

A dissertação apresentada visa o desenho e avaliação de um algoritmo de filiação para uma visão parcial. A análise do algoritmo foca o compromisso existente entre a visão mantida e a eventual violação das restrições acima referidas.

Palavras-chave: editor/assinante, disseminação filtrada, par-a-par, visão parcial

---

Abstract

Nowadays the Internet is an almost infinite source of information. For the user, keeping track of all the information he has interest on is clearly a difficult task when he must sequentially visit all sources and check for new events.

Publish/Subscribe Systems model the interactions between the user as a consumer of information and sources of information. In content based publish/subscribe users are allowed to establish their interests in the form of subscriptions. Centralized solutions are frequent but expensive and limited in terms of scale. More so, most of these solutions do not send notifications to the user when a new event occurs, being up to the user to periodically check for new events.

The Livefeeds Project presents an alternative based on a decentralized peer-to-peer approach. The participants cooperate among themselves in the dissemination of messages taking in account the filters (subscriptions) previously established. This dissemination, also known as content based routing, must ensure that every node that as established interest in a message receives that message. It also has to ensure that nodes do not receive unwanted messages.

The current solution is based on a membership algorithm that creates in each node a complete view of the system. A dissemination algorithm then uses this view to distribute the messages. Although this complete view allows the dissemination algorithm to ensure that the previously stated restrictions hold, the cost of maintaining such view is significant because the system is very dynamic.

In alternative, a partial view membership algorithm may be implemented. In theory, a reduction of the view would linearly translate itself on the cost of membership maintenance. The goal of this dissertation is to present the design and evaluation of a partial membership algorithm. The analysis will focus the compromise between the view reduction and the violation of the previously stated restrictions.

Keywords: Publish/Subscribe, Content Based Routing, Peer-to-Peer, Partial Membership

Conteúdo

Lista de Figuras

Lista de Tabelas

1

Introdução

Inicialmente vista como uma mera interligação de localizações, a infraestrutura a que chamamos de Internet dá hoje suporte a uma vasta gama de aplicações. A rede evoluiu, tornando-se na prática um sinónimo de *World Wide Web* que, sendo uma vasta interligação de conteúdo, demonstra bem a tendência do uso da rede - disponibilização e acesso a informação.

Nos dias de hoje, a quantidade de informação disponível é extremamente volumosa e encontra-se em constante actualização. O utilizador encontra-se desamparado neste universo se pensarmos que tem que procurar a informação que deseja manualmente e fazê-lo periodicamente para se manter ao corrente. A oportunidade era então flagrante no que toca a oferecer ao utilizador facilidade de aceder à informação que este realmente desejava, criando a ideia de fazer a informação vir ao encontro do utilizador de acordo com os seus interesses. Os sistemas editor/assinante surgem no seguimento desta ideia.

Um sistema editor/assinante descreve um paradigma de comunicação entre produtores e consumidores de informação. Estas interacções indirectas são estabelecidas através de tópicos ou subscrições de conteúdo. Os sistemas editor/assinante baseados em subscrições podem basear-se no uso de filtros, que sendo um conjunto de restrições sobre os conteúdos a receber, apresentam-se como os mais interessantes uma vez que possibilitam ao utilizador um maior detalhe na definição dos seus interesses. Geralmente, a distribuição de conteúdos é feita de forma centralizada com base em arquitecturas cliente/servidor. Fontes de conteúdo colocadas num servidor são acedidas pelo cliente quer directamente ou indirectamente através de uma aplicação que serve de intermediário ao processo. Em ambas as situações o servidor simplesmente disponibiliza os conteúdos e a aplicação cliente acede aos mesmos através de um mecanismo *pull-based*. Isto significa que o servidor não faz qualquer tipo de filtragem ou envio de notificações quando um

novo conteúdo é gerado.

Uma arquitectura cliente/servidor *pull-based* será tão escalável quanto a sua capacidade de atender os pedidos que recebe. Numa altura em que a tendência é o aumento do uso desta forma de acesso a informação, é particularmente relevante o estudo de soluções com maior potencial de escalabilidade. Por outro lado, existem as soluções *push-based*. Neste tipo de solução o servidor é responsável por notificar os assinantes quando ocorre um novo evento. Este trabalho envolve conhecer os filtros dos assinantes e determinar a quais enviar as notificações. A este processo chamamos disseminação filtrada. O problema da disseminação filtrada consiste em perante um grupo de participantes, cujos interesses se encontram definidos por meio de subscrições, enviar a todos os participantes as novas mensagens que respeitam as suas subscrições. Este envio deve evitar que os participantes recebam mensagens indesejadas e ao mesmo tempo garantir que estes recebem todas as que desejam. As duas situações são muitas vezes descritas respectivamente como a ausência de falsos positivos e de falsos negativos.

Num sistema de disseminação filtrada centralizado é frequente apelidar-se os servidores de *brokers*, palavra que os descreve como intermediários entre os produtores e consumidores. Ainda assim, isto não significa que um servidor não possa actuar como produtor de informação e ao mesmo tempo actuar como *broker* ou mesmo como consumidor. Um servidor *broker* central conhecerá as subscrições de todo os participantes do sistema e é responsável pela determinação do envio de notificações em função dessas subscrições. Perante tal trabalho, um sistema de servidor *broker* apresenta como limite da sua escalabilidade a sua centralização. O trabalho de análise de filtros e envio de notificações será cada vez maior em função da complexidade dos filtros e do número de participantes do sistema.

Como é típico de soluções baseadas em servidores, os seus problemas podem ser atacados num curto prazo com base no reforço da infraestrutura do servidor, isto é, investindo na capacidade do servidor ou introduzindo novos servidores. No âmbito de múltiplos *brokers*, estes necessitam de colaborar uma vez que o sistema, embora particionado entre servidores, deve ser visto como um todo de forma a garantir que todos participantes recebem as notificações que devem receber. Esta colaboração irá invariavelmente introduzir complexidade no processo de disseminação prejudicando o benefício inerente à existência de múltiplos servidores.

Infraestruturas baseadas em servidores são dispendiosas e algo limitadas pelo que uma alternativa em constante investigação reside na aplicação de princípios *peer-to-peer* a sistemas de disseminação filtrada. Esta alternativa de baixo custo e maior potencial de escalabilidade tira partido das propriedades das redes sobrepostas. O sistema Livefeeds, alvo desta dissertação, insere-se nesta categoria de arquitecturas *peer-to-peer* e apresenta uma possível solução para o problema da disseminação filtrada.

1.1 Motivação

O projecto *Livefeeds* é um exemplo duma aproximação *peer-to-peer* ao problema da disseminação filtrada. Trata-se de um sistema de disseminação de conteúdos do tipo *push-based* que tira partido das propriedades de escalabilidade dos sistemas *peer-to-peer*.

O projecto visa o suporte de sistemas de larga escala em que o trabalho de filtração e disseminação de conteúdo é feito de forma distribuída promovendo a cooperação dos participantes. O sistema preserva o aspecto de servidor (os participantes editores de conteúdo), no entanto alivia estes servidores do *polling* constante a que hoje se encontram sujeitos.

Facilmente podemos observar o benefício e motivação de um sistema deste tipo quando pensarmos num caso em que temos um número elevado de utilizadores que acedem frequentemente a um servidor. O sistema Livefeeds permite reduzir drasticamente o número de utilizadores que acedem directamente ao servidor, sendo que os nós que realmente acedem ao servidor podem tornar-se raízes de árvores de disseminação entre os participantes do sistema.

Para além das motivações de potencial de escalabilidade e baixo custo associadas a uma arquitectura *peer-to-peer*, é possível também explorar estas ligações entre participantes. Por exemplo, permitindo que participantes tomem conhecimento através de outros de fontes de interesse em comum. Estas funcionalidades, de cariz mais social e cada vez mais presentes, são mais naturais numa arquitectura *peer-to-peer* em contrapartida ao modelo actual em que os utilizadores actuam de forma independente em acessos isolados a servidores.

1.1.1 Os Algoritmos Livefeeds

Actualmente a solução descrita pelo projecto Livefeeds envolve dois algoritmos. Um algoritmo de filiação e um algoritmo de disseminação filtrada.

1.1.1.1 Algoritmo de Filiação

O algoritmo de filiação do Livefeeds tem por objectivo produzir em cada nó o conhecimento dos filtros de todos os participantes do sistema. O algoritmo baseia-se na agregação e disseminação de filtros dos participantes. Uma vez que o conhecimento necessário diz respeito a todos os nós do sistema dizemos que este algoritmo produz uma visão completa.

A agregação dos filtros é feita através de nós sequenciadores que são responsáveis por partições do espaço de identificadores. Esta agregação é feita durante um determinado período tempo sendo depois disseminada para o resto dos participantes. Para colmatar eventuais falhas do protocolo anterior, uma vez que se trata de um envio *best effort*, é usado um mecanismo epidémico através do qual os nós podem corrigir as suas visões do sistema com base nas dos seus parceiros.

1.1.1.2 Algoritmo de Disseminação

O segundo algoritmo diz respeito à disseminação de conteúdos no sistema. Como referido, esta é feita de forma distribuída pelos participantes usando a informação que existe em cada nó como resultado do processo de filiação. Mediante o conhecimento dos filtros de todos os participantes, cada nó determina exactamente quais os nós cujos filtros são compatíveis com a mensagem a encaminhar, garantindo que a mensagem não é entregue a um nó cujo filtro não a aceite (falso positivo).

De forma a distribuir a carga da disseminação esta é repartida em intervalos. Isto é, um nó ao enviar uma mensagem para outro nó envia também um subintervalo em que será da responsabilidade desse segundo nó disseminar essa mensagem, dando origem a processo de disseminação em árvore em que os intervalos vão ficando cada vez mais pequenos até terminar a disseminação.

1.1.2 Limitações do actual Algoritmo de Filiação

Apesar de diversas optimizações o algoritmo de filiação com visão completa do sistema é relativamente caro, um custo mensurável em termos de largura de banda. Possuir uma visão completa de um sistema altamente dinâmico resulta em custos significativos associados à entrada de cada participante.

O elevado dinamismo de um sistema, muitas vezes denominado de *churn*, é caracterizado pela taxa de chegadas de novos participantes e a duração da sua permanência no sistema. Uma elevada taxa de chegadas associada a um tempo de sessão baixo dissolvem o benefício de uma visão completa do sistema uma vez que é necessário um grande esforço para manter essa visão.

Também a agravar o problema do dinamismo do sistema encontra-se a complexidade dos filtros. Caso os filtros possuam uma dimensão elevada a sua propagação por todos os participantes terá custos, em termos de largura de banda, mais elevados em função dessa mesma dimensão.

Com o intuito de tornar o sistema menos dispendioso face a um alto dinamismo e uma vez que um dos principais objectivos do projecto Livefeeds é acomodar elevadas taxas de chegada de participantes, é de todo o interesse explorar alternativas. Uma dessas alternativas encontra-se numa solução com visão parcial que, mantendo uma visão parcial do sistema em cada nó, reduz a largura de banda necessária permitindo que os custos sejam reduzidos linearmente em relação à redução do estado mantido.

1.2 Objectivos e Contribuições

O alto dinamismo no sistema Livefeeds é um factor limitante uma vez que temos um elevado número de entradas para disseminar entre todos os participantes de forma a obter uma visão completa. Uma solução para reduzir o custo associado ao algoritmo

de filiação é então reduzir a visão que os participantes têm do sistema. Passamos assim duma visão completa para uma visão parcial.

Consideremos que um participante é definido pelo seu identificador, o seu endereço e o seu filtro. Numa primeira abordagem, sabendo que o que consome mais largura de banda é a disseminação de filtros, podemos limitar o conhecimento dos filtros a uma vizinhança do nó. O nó mantém no entanto conhecimento global dos identificadores e endereços.

Esta dissertação visa adaptar o algoritmo de filiação para uma nova versão com visibilidade parcial. Este processo envolve uma alteração significativa daquele que era o paradigma da existência de uma visibilidade completa. São descritas estas alterações e é feita uma análise do seu impacto nos custos de largura de banda através de estimativas que permitem uma visão preliminar dum sistema antes da sua implementação. Derivada da alteração do algoritmo de filiação encontra-se a necessidade de adaptação do algoritmo de disseminação filtrada. Resultante da perca de visibilidade, a possível introdução de falsos positivos é um aspecto crucial que avaliamos em grande detalhe nesta dissertação.

Esta dissertação contribui para o desenvolvimento de uma alternativa no sistema Livefeeds, tendo resultado na implementação de um protótipo simulado que é base da validação experimental e que, juntamente com uma análise detalhada, visa demonstrar a viabilidade desta alternativa.

1.3 Estrutura do Documento

No capítulo 2 é apresentado o trabalho relacionado nos âmbitos de *Content Based Routing*, redes sobrepostas e a versão existente do sistema Livefeeds. De seguida, no capítulo 3, descrevemos a solução proposta juntamente com uma análise detalhada da mesma. No capítulo 4 descrevemos o ambiente de simulação, o protótipo e os resultados experimentais obtidos. Finalmente, no capítulo 5 apresentamos as conclusões finais bem como o trabalho futuro a realizar no âmbito deste tema.

2

Trabalho Relacionado

2.1 *Content Based Routing*

Desde o seu desenho inicial até aos dias de hoje a vasta rede de redes que chamamos *Internet* consiste numa interligação de localizações via canais de comunicação. Todo o encaminhamento oferecido pela rede é focado na localização. Isto é, quando queremos aceder a determinado conteúdo da rede é necessário especificar a sua localização. Analogamente, quando queremos enviar algo, temos de definir concretamente a localização de destino, localizações que são identificadas univocamente por endereços.

Apesar da rede disponibilizar encaminhamento baseado em endereços, a maior parte do uso da rede consiste no acesso a conteúdos e, na maior parte dos casos, quando queremos aceder a conteúdos não estamos interessados na sua localização. *Content Based Routing* consiste em realizar encaminhamento não dependente das localizações mas sim do conteúdo.

Um modelo de comunicação que tem como objectivo desagregar o acesso a conteúdos das dimensões do espaço e tempo, possuindo uma grande vertente de *Content Based Routing*, encontra-se no paradigma editor/assinante.

2.1.1 Editor/Assinante

O modelo editor/assinante, amplamente conhecido pelo termo inglês *publish/subscribe*, trata-se de um paradigma de comunicação em que nós, editores, publicam eventos que se dividem em classes. Outros nós, assinantes, estabelecem por sua vez o seu interesse nas classes de eventos que querem receber (subscrições). Quando um editor dá origem a um evento, os assinantes interessados devem ser notificados. O envio destas notificações não

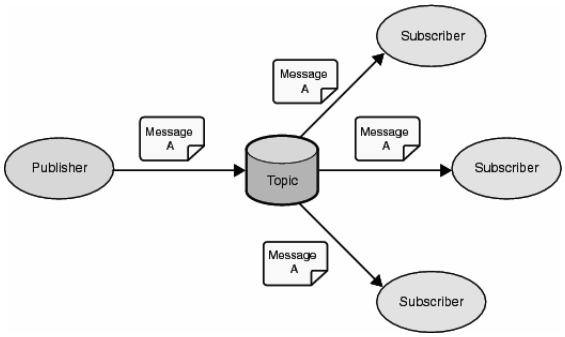

Figura 2.1: Editor/Assinante baseado em tópicos.

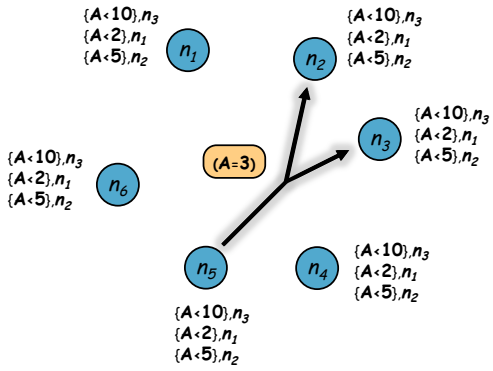

Figura 2.2: *Content Based Routing* Editor/Assinante baseado no conteúdo. [20]

é feito de forma directa mas sim de forma implícita ao grupo formado pelos assinantes.

Existem duas vertentes mais comuns de sistemas editor/assinante. Sistemas baseados em tópicos e sistemas baseados no conteúdo.

Editor/Assinante baseado em Tópicos Consiste em ter mensagens associadas a tópicos. Os participantes publicam eventos no tópico que é por sua vez subscrito pelos nós assinantes. Quando um evento ocorre todos os nós associados ao tópico são notificados.

Editor/Assinante baseado no Conteúdo Consiste em permitir que os nós estabeleçam os seus interesses por meio de subscrições referentes às diferentes classes de conteúdo e recebam assim as mensagens cujo conteúdo respeita a sua subscrição.

Em sistemas baseados no conteúdo, as mensagens não se encontram endereçadas e, como tal, não são dependentes da localização. Após publicada uma mensagem, é geralmente da responsabilidade de uma camada sob a aplicação, fazer com que esta chegue a todos nós interessados e apenas aos interessados. Gerando assim respectivamente, nenhum falso negativo e nenhum falso positivo.

As mais conhecidas utilizações destes sistemas encontram-se por exemplo na disseminação de *RSS feeds* e um grande número de sistemas de notificação de eventos noticiosos.

Os sistemas editor/assinante podem seguir uma arquitectura cliente/servidor ou adoptar uma solução descentralizada *peer-to-peer*. Numa arquitectura cliente/servidor é previsível que o servidor se torne num ponto de limitação do sistema, uma vez que é responsável por todo o processo de determinação de que nós devem ser notificados bem como pela gestão de global do sistema. Considerando que o processo se torna cada vez mais complexo, podendo demorar mais tempo do que aquele que é exigido, podemos tentar distribuir o processo por vários servidores. Em alternativa podemos ainda desenhar uma aproximação *peer-to-peer* puramente descentralizada. Uma solução deste tipo é especialmente relevante quando não existem incentivos financeiros para criar uma infraestrutura de múltiplos servidores dedicados.

No caso de abordagens *peer-to-peer*, sistemas deste género são geralmente implementados sobre redes sobrepostas tirando partido das propriedades de comunicação, escalabilidade e resistência a falhas que estas procuram oferecer [33, 1].

2.2 Redes Sobrepostas

Muitas das soluções propostas em termos de sistemas editor/assinante assentam no uso de redes sobrepostas. Estas oferecem uma infraestrutura com propriedades de organização, escalabilidade, encaminhamento e tolerância a falhas que são úteis ao desenho de sistemas editor/assinante.

Uma rede sobreposta ou rede *overlay* é uma rede virtual de nós e canais (*links*) lógicos que, ao ser implementada sobre as redes físicas actuais, permite reutilizar as infraestruturas e protocolos já existentes. A reutilização é uma grande vantagem e uma necessidade quando a infraestrutura existente, baseada no protocolo IP, começa a ser uma vítima do seu próprio sucesso exibindo elevados custos de substituição que são proporcionais à escala da sua utilização.

As redes sobrepostas apresentam-se como uma alternativa a essa substituição em larga escala, permitindo rápida implementação de novos protocolos sobre a pilha existente. No entanto, estas incorrem num maior *overhead* devido à existência de uma camada adicional (*middleware*).

É preciso notar no entanto que os *links* lógicos desta rede virtual podem corresponder a vários *links* físicos e à repetida utilização destes, por isso, um caminho óptimo ao nível da rede sobreposta pode não ser o caminho óptimo real ao nível da rede física que a suporta (Figura 2.3).

Pondo de parte a optimalidade do caminho real, a problemática inerente a estas redes consiste em saber como realizar encaminhamentos entre os seus participantes. A forma de encaminhar uma mensagem numa rede sobreposta depende de como esta está organizada e das interligações entre os seus participantes. Numa aproximação os participantes

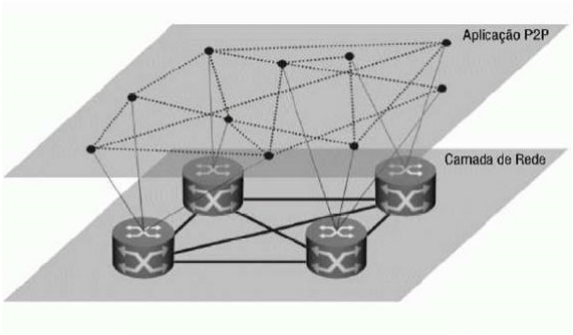

Figura 2.3: Rede física e Rede Sobreposta

ligam-se sem qualquer regra formando estruturas aleatórias. A este tipo de redes chamamos redes não estruturadas. Noutra aproximação, os nós formam estruturas virtuais de acordo com determinada regra, sendo estas denominadas de redes estruturadas.

As redes sobrepostas encontram-se na base dos sistemas *peer-to-peer* que, tirando partido da maior capacidade de processamento e memória dos nós participantes (em comparação com os *routers* convencionais) permitem a implementação de mecanismos não oferecidos pelas camadas inferiores.

Um exemplo de implementação de um mecanismo que se provou complexo ao nível do protocolo IP é o mecanismo de comunicação *multicast*. A falta deste ao nível IP fez com que a sua implementação fosse explorada ao nível aplicacional.

Sistemas *peer-to-peer* assentes em redes sobrepostas são, por exemplo, usados para a partilha de ficheiros, distribuição de conteúdos *publish/subscribe* [7, 29], comunicação em grupo (*multicast*) [2, 6], armazenamento de dados [13], entre outros. A sua natureza descentralizada permite implementar infraestruturas de suporte aos vários tipos de aplicações com facilidade e com baixos custos. No entanto, uma desvantagem desta aproximação reside na própria descentralização do sistema que torna difícil aplicar modelos de negócio tradicionais a aplicações que usam este tipo de redes.

2.2.1 Redes Não Estruturadas

2.2.1.1 Organização

Redes não estruturadas são um tipo de redes sobrepostas em que os nós tipicamente formam ligações de forma aleatória não obedecendo a nenhuma regra pré-definida.

O sistema *Gnutella* [26] é um exemplo deste tipo de redes, e foi uma das primeiras abordagens às redes sobrepostas num sistema totalmente descentralizado. Trata-se de um sistema de partilha de ficheiros, uma popular aplicação de redes sobrepostas. Aplicações desta natureza requerem pesquisas (encaminhamentos), no entanto a dificuldade de encontrar dado objecto ou encaminhar uma mensagem para dado destinatário é óbvia já que a rede é aleatória.

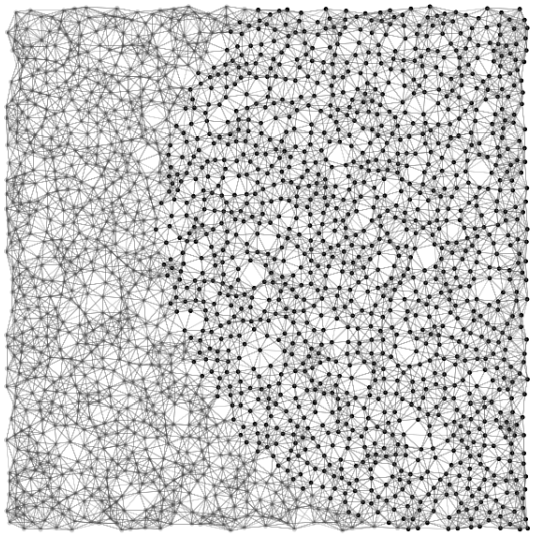

Figura 2.4: *flooding* numa rede não estruturada

2.2.1.2 Manutenção

Uma das principais vantagens das redes não estruturadas é que não existindo restrições (excepto possivelmente em número) em relação aos parceiros que um nó se pode ligar, sendo estas aleatórias, uma rede não estruturada requer um reduzido esforço de manutenção. Isto significa que as redes não estruturadas lidam trivialmente com o dinamismo verificado na maioria dos cenários de utilização.

2.2.1.3 Encaminhamento

Mediante uma estrutura aleatória, os principais algoritmos de encaminhamento nestas redes são baseados em *flooding* ou *random walks*, sendo por isso não determinísticos.

Flooding *Flooding* consiste em enviar uma mensagem para todos os vizinhos conhecidos, que por sua vez repetem o processo para os seus e assim por diante. O uso de mecanismo de *flooding* provoca um *overhead* generalizado. A propagação duma mensagem requer sempre filtragem ao nível da recepção e por isso aplicações do tipo *publish/subscribe* são contra a natureza destas redes. Qualquer mecanismo de *flooding* é acompanhado de um mecanismo de detecção de duplicados e na grande maioria dos casos limitado através de um *TTL*. Esta limitação permite limitar o *overhead* provocado pelas pesquisas. No entanto, pode diminuir o sucesso das mesmas.

Random Walk Um *random walk* consiste em enviar uma mensagem que percorre a rede aleatoriamente com a esperança que esta chegue eventualmente ao seu destino. Na maior parte dos casos vários *walkers* são usados para maximizar a probabilidade de sucesso ao mesmo tempo que o número de *hops* que podem efectuar é limitado (de forma a não degenerar para uma inundação).

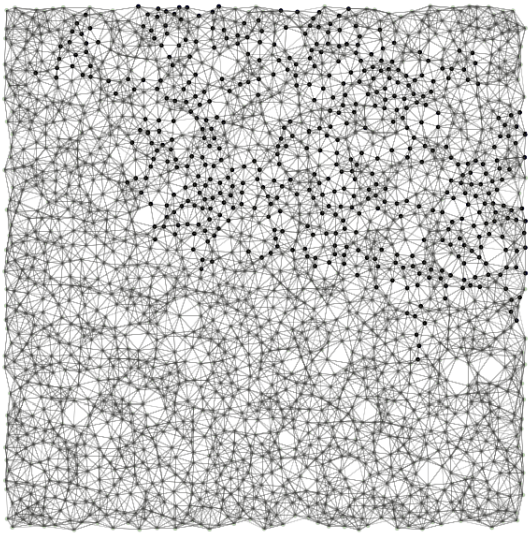

Figura 2.5: *random walking* numa rede não estruturada

Pesquisas Informadas Os métodos de pesquisa/encaminhamento descritos acima consistem em pesquisas cegas. Isto é, a decisão de encaminhar para certo vizinho é feita ao acaso e não tem por base nenhuma informação. Por oposição a estas, existe um tipo de pesquisa chamadas pesquisas informadas. Neste caso, um nó pode ter já conhecimento que para certa pesquisa o melhor caminho a seguir é determinado vizinho e por isso a sua decisão é influenciada pela informação que detém.

Os nós participantes podem manter tabelas de *routing* ou utilizar outro tipo de heurísticas. Note-se que o facto de existirem tabelas de *routing* não significa que a rede seja estruturada uma vez que a escolha de parceiros continua a ser aleatória. Um exemplo de sistema que usa tabelas de *routing* para realizar pesquisas informadas é o sistema Freenet [10].

A informação pode ser eficientemente guardada por exemplo usando *bloom filters* atenuados [14]. Um *bloom filter* é uma estrutura de dados altamente compacta baseada em *hashing*. Consiste num vector de bits que são activados consoante o resultado do *hashing*. Devido à possibilidade de colisões um *bloom filter* é uma estrutura probabilística em que falsos positivos podem acontecer.

No contexto das redes não estruturadas este pode ser usado para guardar informação numa aproximação de *bloom filters* atenuados, em que existe um *bloom filter* para cada nível de distância de nós (um hop, dois hops, e assim sucessivamente até ao limite imposto).

Os mecanismos de encaminhamento descritos introduzem redundância o que, como

	0	1	2	3	4	5	6	7
First Hop	1	1	0	1	0	1	1	0
Second Hop	1	0	0	0	1	0	0	1
Third Hop	1	1	1	1	0	0	1	0

Figura 2.6: Exemplo de um *bloom filter* atenuado

referido, produz *overhead*. No entanto esta mesma redundância confere a estes encaminhamentos uma maior robustez dado que a mensagem segue múltiplos caminhos resistindo a falhas de nós.

Actualmente as redes sobrepostas não estruturadas evoluíram para sistemas *multi-tier*. Nestes, nós apelidados de super-nós formam redes ao nível superior, enquanto os outros participantes ligam-se a esses nós [31].

Ainda assim, embora este tipo de redes permita um certo anonimato entre os participantes e sejam fáceis de implementar, as redes não estruturadas mostraram-se limitadas em relação à escala já que o *overhead* dos protocolos usados é por natureza muito elevado e quando limitados diminuem o sucesso dos encaminhamentos.

2.2.2 Redes Estruturadas

2.2.2.1 Organização

Nas redes estruturadas, por oposição às não estruturadas, os nós participantes ligam-se segundo regras bem definidas formando estruturas lógicas bem organizadas tendo em conta identificadores globais de cada nó.

A investigação nesta área foca o compromisso entre o estado mantido pelos nós, a robustez do sistema, a complexidade das pesquisas e, a largura de banda utilizada por cada participante individualmente e globalmente em termos da distribuição da carga de trabalho realizada pelo sistema.

Um dos usos mais comuns de tipo de redes sobrepostas é a implementação de *distributed hash tables*, um sistema distribuído mapeando dados a identificadores e com uma política de replicação e noção de tolerância a falhas.

Distributed Hash Tables *Distributed Hash Tables* são um tipo de sistema distribuído de armazenamento de pares <*Key,Value*> que distribuem este armazenamento o mais uniformemente possível entre os participantes e fornecem um mecanismo de *lookup*. O sistema é também responsável por manter a sua organização e coerência na presença de alterações da rede.

São exemplos clássicos deste tipo de redes os sistemas *Chord*, *CAN*, *Tapestry* e *Pastry* [11, 28, 25, 35]. Particularmente no *Chord* e no *Pastry* o esforço centrou-se em manter um

estado relativamente reduzido de informação em cada nó ($\log(N)$) e ainda assim obter pesquisas também de complexidade logarítmica.

Após as aproximações citadas, e mediante a evolução do *hardware* e a observação da evolução das aplicações *peer-to-peer* na prática, começou-se a questionar a necessidade *dht's multi-hop* nas aplicações reais, tendo surgido várias *dht's one-hop* ou n*-hop* [27, 17, 16, 18] sendo n um valor reduzido à custa de um maior estado guardado nos nós. O sistema *Livefeeds*, alvo desta dissertação, insere-se nesta segunda corrente.

2.2.2.2 Manutenção

Devido a sua estrutura condicionada, as redes estruturadas necessitam de se organizar constantemente em função das entradas e saídas e nós participantes (*churn*). O esforço despendido para lidar com o dinamismo do sistema traduz-se geralmente no consumo de largura de banda, dado que, as alterações têm de ser propagadas pela rede. Quanto maior o grau dos nós maior largura de banda será necessária para manter informação sobre os nós conhecidos. Por outro lado, se este número for muito pequeno, significa que o número de nós candidatos a serem vizinhos de um nó é muito restritivo. E por isso, o esforço necessário para encontrar e manter este nós é significativo num sistema altamente dinâmico.

Ainda assim, essa mesma estruturação pode ser explorada e mediante mecanismo apropriados, estudos demonstram que estas podem lidar com esse problema de forma eficiente [5, 4].

2.2.2.3 Encaminhamento

A estruturação da rede implica que podemos desenvolver mecanismos de encaminhamento determinísticos. Isto é, dado a estruturação da rede em função de identificadores, um encaminhamento consiste em entregar uma mensagem a um identificador específico. Os nós participantes, ao contrário das redes não estruturadas, não escolhem aleatoriamente a que vizinho entregar uma mensagem. De acordo com a informação que detêm a dado momento, escolhem aquele que sabem ser o caminho para alcançar o destino. Desta forma evita-se o *overhead* das redes não estruturadas. O caminho percorrido por uma mensagem depende obviamente dos participantes conhecidos por cada nó. Quantos mais nós forem conhecidos menos nós intermediários são contactados.

Enquanto nas redes não estruturadas os encaminhamentos eram redundantes e por isso inerentemente tolerantes a falhas, no caso das redes sobrepostas o sucesso dos encaminhamentos depende da correcção das tabelas de *routing* em cada nó. A mensagem segue um caminho bem determinado e o envio desta para um nó que na verdade se encontra *offline* resulta na falha do envio ou na tentativa de corrigir as tabelas de *routing* antes de prosseguir com o encaminhamento.

Este encaminhamento para um identificador encontra-se bem definido e o custo esperado é relativamente baixo. No entanto, as *dht's* apresentam limitações caso o encaminhamento necessário seja, por exemplo, uma pesquisa por atributos ou a distribuição de conteúdo. Para estes encaminhamentos a aproximação *unicast* oferecida pelas *dht's* é limitadora e envolve maior esforço ao nível aplicacional ou, numa camada intermédia, para a implementação desses mesmos encaminhamentos.

Uma vasta gama de aplicações tira partido das capacidades de encaminhamento, auto-organização e escalabilidade destas redes. Estas oferecem, por exemplo, uma boa plataforma para o desenvolvimento de aplicações do tipo *publish/subscribe* [7, 29]. Neste género de aplicação os participantes organizam-se em grupos implícitos ou explícitos sob a forma de subscrições a nós produtores. A disseminação de mensagens dentro de estes grupos consiste numa comunicação *multicast* que é implementada sobre a comunicação *unicast* oferecida pela rede *overlay*.

2.2.3 Visão geral de algumas abordagens ao desenho de Redes Estruturadas

2.2.3.1 Chord

O *chord* trata-se de uma *dht* que implementa um protocolo de *lookup* distribuído. O *chord* tem por base um desenho em anel, mapeando chaves aos identificadores dos nós e com cada nó a conhecer $\log(n)$ outros nós (*fingers*).

Permitindo pesquisas *multi-hop* de complexidade $O(\log(n))$, os pontos que o distinguiram dos algoritmos prévios foram a sua simplicidade e a possibilidade de provar tanto a sua correcção como o seu desempenho [11].

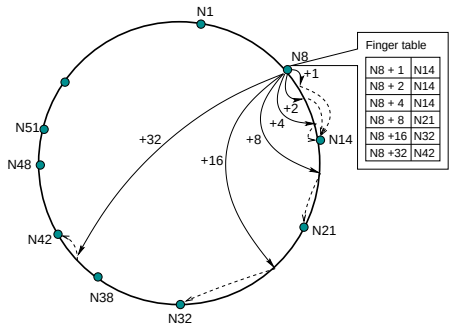

Figura 2.7: Estrutura Lógica do *chord* e vista dos *fingers* [11].

Os eventos de entrada de nós podem sempre causar situações de incoerência e as pesquisas irão falhar. O *chord* delega o tratamento desse caso para a aplicação já que se consideram que os tempos de convergência são reduzidos pelo que a aplicação pode reiterar a pesquisa. A entrada de nós no sistema envolve o conhecimento prévio de um nó participante. O novo nó envia um pedido *join* para esse nó. Através deste pedido o

novo nó fica a saber qual o seu sucessor. A existência de um mecanismo periódico de *probing* faz com que o sucessor do novo nó tome conhecimento da existência deste.

De forma a lidar com a falha de nós, cada nó mantém uma lista de r sucessores, sendo r aproximadamente $2\log_2(N)$. Deste modo, quando o primeiro falha pode ser substituído pela segunda entrada. É muito pouco provável que todos os nós desta lista falhem.

O sistema baseia o seu balanceamento de carga na distribuição da função de *hashing* que distribui as chaves pelos nós e assenta a sua escalabilidade no facto de possuir um desenho logarítmico.

2.2.3.2 Pastry

O desenho do *Pastry* à semelhança do *chord* assenta nos princípios de desenho de *dht's*. Trata-se de um desenho descentralizado e tolerante a falhas com *routing* baseado em prefixos baseado nas redes de Plaxton [24] e com a particularidade de o protocolo de *routing* poder ser adaptado a heurísticas tendo em conta a localidade dos nós. Como por exemplo o número de *IP hops* entre outras possíveis métricas.

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	**1**	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	**2**	10-3-23302
102-0-0230	102-1-1302	102-2-2302	**3**
1023-0-322	1023-1-000	1023-2-121	**3**
10233-0-01	**1**	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figura 2.8: Estado guardado num hipotético nó no sistema *Pastry* [28].

Neste sistema cada nó mantém conhecimento de um conjunto de nós cujos identificadores são iguais ao próprio no maior número possível de dígitos (*leaf set*), uma tabela de *routing* em que cada entrada consiste num nó que é usado para alcançar o prefixo que esse nó representa e um conjunto de nós em que o IP é próximo do nó local (*neighborhood set*).

O encaminhamento duma mensagem é feito através dum algoritmo do tipo *longest prefix match* em relação à tabela de *routing*.

À semelhança de outros sistemas, um novo nó (X) junta-se ao sistema através de um nó participante previamente conhecido (A). X pede a A para encaminhar uma mensagem para *id(X)*, ficando a conhecer Z, o nó mais próximo de *id(X)*. Através de Z, X obtém o *leaf set* e através de A o *neighborhood set*. A tabela de *routing*, por sua vez, é formada pelos nós contactados no processo de *routing* de A a Z.

O *Pastry* permite pesquisas em $\lceil \log_{2^b}(N) \rceil$ passos (sendo b um parâmetro de configuração) [28].

2.2.3.3 Tapestry

O *Tapestry* é juntamente com o *Chord*, *Pastry* e *CAN* um sistema da primeira geração de *dht's*. O *Tapestry* apresenta um desenho semelhante ao *Pastry* baseado em redes de Plaxton [24], usando um algoritmo de *routing* baseado, neste caso, em sufixos.

Os mecanismos de inserção de nós são semelhantes, envolvendo no *Tapestry* também, o *routing* duma mensagem para o próprio nó que pretende juntar-se ao sistema.

Enquanto que no *Pastry* a falha de nós é resolvida recorrendo ao *leaf set* ou ao *neighboorhood set* no *Tapestry* estes não existem. Na verdade para cada entrada na tabela de *routing*, existem três nós conhecidos. Ordenados de acordo com uma métrica de proximidade (latência), funcionam como *backup* os dois para além do activo. De forma a minimizar o custo de inserções de nós e como é expectável que um nó que falhe recupere, nós que falhem não são prontamente removidos, mas sim considerados inactivos, sendo lhes dado um período de tempo para recuperarem.

No *Tapestry* ao contrário do *Pastry* não são criadas várias réplicas de um objecto na sua inserção. Em vez disso, durante a inserção, ao *ID* do objecto é adicionado uma sequência de valores de forma que, ao usar *hashing*, são gerados vários *ID's* em cujos nós responsáveis vão possuir informação de como localizar o objecto, criando redundância ao alcançar o objecto.

O *Tapestry* oferece *lookups* em aproximadamente $\log_b(N)$ *hops*, sendo b a base do espaço de identificadores.

2.2.3.4 CAN

O *CAN*, siginificando *Content Addressable Network*, apresenta um modelo em que os nós são mapeados num espaço de n dimensões através de *hashing*. Tirando partido do mapeamento de nós em coordenadas, o algoritmo de *routing* consiste em encaminhar para o nó vizinho com as coordenadas mais próximas do destino.

Cada nó é responsável pela área à volta das suas coordenadas a que chamamos a sua zona (Figura 2.9).

Para se juntar ao sistema um nó escolhe um ponto aleatório do espaço para o qual envia uma mensagem. O nó responsável pela zona recebe o pedido, decide se divide a zona em dois atribuindo metade ao novo nó. Caso o nó recuse dividir, o nó que pretende juntar-se ao sistema escolhe outro ponto aleatório.

Quando um nó sai do sistema é necessário encontrar um responsável pela zona deixada pela o nó que saiu. Para isso o sistema testa a zona quanto à capacidade de se fundir com as zonas dos vizinhos, fazendo a fusão caso possível. Caso contrário, a responsabilidade é atribuída ao nó vizinho com a menor zona.

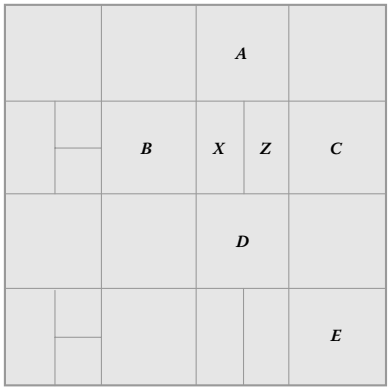

Figura 2.9: Espaço $2d$ no sistema *CAN* [32].

A falha de nós é tratada por um mecanismo semelhante ao das saídas. Os nós vizinhos enviam periodicamente *updates* entre si, a falha da recepção em tempo esperado indica a falha do nó correspondente. Detectada a falha de um nó X num nó Z, Z assume temporariamente o controlo da zona de X.

2.2.3.5 Koorde

O *Koorde*[19] é uma *dht* de segunda geração baseada no *Chord*. Os autores deste sistema propõem uma estruturação com base em grafos de Bruijn (a figura 2.10 mostra um exemplo deste tipo de grafo) permitindo pesquisas logarítmicas com apenas dois nós conhecidos por cada nó. No entanto, conhecendo apenas dois nós a tolerância a falhas do sistema é consideravelmente menor pelo que os autores também defendem que conhecendo $\log(n)$ é possível obter no *Koorde* pesquisas com $O(\log(n)/\log(\log(n)))$ passos.

O *Koorde* e o *Viceroy* representam aproximações que sacrificam tolerância a falhas e balanceamento de carga em favor da sua optimalidade mas que oferecem ainda assim margem de manobra para encontrar um meio termo.

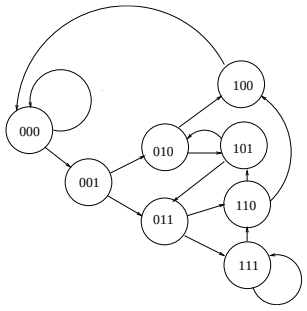

Figura 2.10: Grafo de Bruijn, $b = 3$ [19].

2.2.3.6 Viceroy

O *viceroy* surge, por oposição às *dht's* de grau variável, como uma *dht* de grau fixo. De forma semelhante aos exemplos anteriores o *viceroy* usa *hashing* para mapear os nós num espaço de identificadores entre zero e um. Os nós organizam-se num anel global (predecessor e sucessor) e, adicionalmente, os nós são distribuídos por níveis formando a estrutura visível na figura 2.11. Esta estrutura implica que um nó do sistema tem grau sete.

A vantagem desta aproximação é que a reestruturação provocada pela entrada ou saída de um nó do sistema é mínima, afectando apenas o estado de um número muito restrito de nós e tornando o sistema tolerante a um alto dinamismo. No entanto, note-se que aqui se referem saídas anunciadas e não falhas de nós.

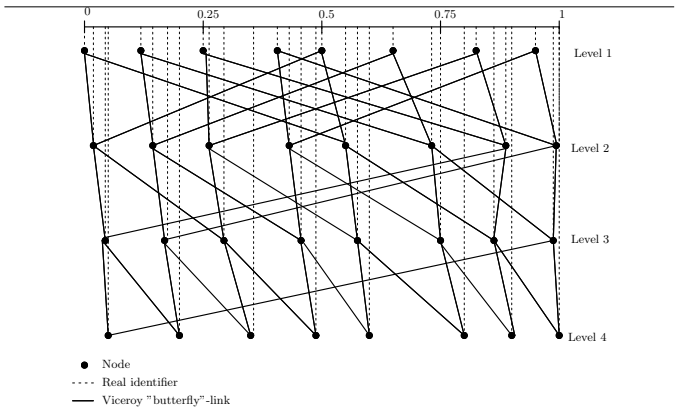

Figura 2.11: Estrutura Lógica dos vários níveis no sistema *Viceroy* [21].

2.2.3.7 Efficient Routing for peer-to-peer Overlays

No artigo [16] os autores propõem um sistema de visão total com vista a obter encaminhamentos *one-hop* à custa de um maior estado guardado no nó. O sistema consiste numa rede sobreposta em que cada nó possui um identificador e estes encontram-se uniformemente distribuídos pelo domínio de identificadores. Os nós estão organizados de forma circular e cada um possui assim um antecessor e um sucessor. O domínio de identificadores é dividido em intervalos (*slices*) e subintervalos (*units*) existindo nós com responsabilidades acrescidas dentro de cada um destes intervalos (*slice leaders* e *unit leaders*). Esta hierarquia é então usada como árvore de disseminação de eventos com vista a manter em cada nó uma visão completa dos sistema que seja coerente.

Esta aproximação suporta facilmente sistemas na ordem de 10^5 nós, no entanto, para sistemas de maior dimensão a largura de banda necessária a cada nó pode ser significativa pelo que os autores propõem uma variação *two-hop*. A visão de cada nó é restringida com cada nó a conhecer todos os nós presentes na sua *slice* e em vez de conhecer todos os nós exteriores, conhece um nó em cada *slice*. A pesquisa consiste agora em enviar o pedido para o nó conhecido na *slice* de destino que por sua vez encaminhará o pedido ao nó na sua *slice*.

2.2.3.8 Kelips

O *Kelips* consiste num sistema que à custa de maior uso de memória nos nós ($O(\sqrt{n})$) e de *overhead* secundário, permite pesquisas de complexidade temporal $O(1)$.

No *Kelips* os nós são distribuídos em k grupos, cuja filiação é determinada através de uma função de *hash* que mapeia o endereço do nó para um inteiro no intervalo $[0, k-1]$. Em cada nó é mantida uma vista do grupo a que o nó pertence, uma vista esparsa de nós presentes em cada um dos outros grupos (*contacts*) e uma tabela consistindo em tuplos descrevendo ficheiros que se encontram presentes em nós pertencentes ao mesmo grupo que o nó (*filetuples*). Estas vistas são periodicamente refrescadas com base num mecanismo de *heartbeat*. Através deste são detectadas alterações na rede. Estas alterações são disseminadas intra-grupo através de um mecanismo epidémico, pesado com base em estimativas de *rtt*, que dissemina as saídas, entradas e novos ficheiros. De forma a propagar informação entre diferentes grupos são escolhidos também alguns nós da vista dos nós exteriores ao grupo para serem alvo da epidemia.

Sistema	Grau	*lookup*
Chord	$\log(n)$	$\log(n)$
Pastry	$\log(n)$	$\lceil \log_{2^b}(N) \rceil$
Tapestry	$\log_b(n)$	$\log_b(n)$
CAN	$2d$	$dN^{1/d}$
Koorde	2	$\log(n)$
Koorde	$\log(n)$	$\log(n)/\log(\log(n))$
Viceroy	7	$\log(n)$
One-hop DHT	n	1
Two-hop DHT	$(n/k + (k-1))$	2
Kelips	$\sqrt{n}$	1

Tabela 2.1: Comparação entre os vários sistemas apresentados

2.3 Abordagens conhecidas dentro do Paradigma Editor/assinante

2.3.1 Servidores (*Brokers*)

Um sistema editor/assinante pode ser implementado usando uma aproximação Cliente/Servidor ou *peer-to-peer*. Num modelo Cliente/Servidor os participantes do sistema dependem de servidores denominados *brokers* que medeiam as interacções indirectas entre editores e assinantes.

O uso de *brokers* significa que estes são responsáveis por determinar que nós devem ser notificados quando um novo evento é gerado por um nó editor. No caso de existir um número elevado de nós ligados a um *broker* em particular, o esforço requerido para determina de acordo com as subscrições que nós devem ser notificados, torna-se significativo e é um ponto crítico em termos da escalabilidade desta aproximação.

De forma a permitir elevados números de participantes podem ser usados vários servidores. Esta aproximação tem o intuito de dividir a carga entre os vários servidores. Mediante vários servidores é necessário que estes encaminhem notificações entre eles, actuando como *peers*.

Push-based brokers consiste no paradigma de utilização de *brokers* em que é da responsabilidade dos *brokers* fazer chegar as notificações aos assinantes. Geralmente, usando alguma forma de *multicast*.

Pull-based brokers consiste no paradigma inverso ao anterior. Os *brokers* determinam as notificações e guardam-nas ficando disponíveis para que sejam mais tarde acedidas pelos assinantes.

Uma solução baseada em *brokers* é conceptualmente simples e a sua eficiência pode ser promovida através do uso de redes de *brokers*. No entanto, estruturas baseadas em servidores requerem um grande esforço financeiro. Em alternativa, na ausência de tais incentivos, sistemas *peer-to-peer* podem ser usados.

2.3.2 Tabelas de Encaminhamento baseadas em Subscrições

Num sistema descentralizado do tipo *peer-to-peer* os nós podem ser simultaneamente editores, assinantes e *routers* de notificações. Neste tipo de sistemas coloca-se o problema de como sabem os nós para onde encaminhar as notificações. De forma a resolver esse problema criam-se árvores e limitam-se as mesmas em função das subscrições dos utilizadores.

Uma possível solução é o uso de tabelas baseadas em subscrições. Estas tabelas são preenchidas usando anúncios feitos pelos nós relativos às suas subscrições. Este método, chamado aprendizagem pelo caminho inverso, consiste em inferir que dado nó X, com a subscrição S, é alcançável pelo canal por onde chegou o anúncio de X. Desta forma, quando uma notificação é recebida são usadas as tabelas para determinar quais as subscrições compatíveis com a notificação e assim, quais os canais por onde esta deve ser propagada.

A vantagem desta aproximação é, que estando as tabelas correctas, evita a propagação desnecessária de notificações. Por outro lado é necessário manter as tabelas de encaminhamento o que envolve custos, nomeadamente em termos de largura de banda.

Esta aproximação pode ainda ser alvo de melhoramentos que visam reduzir o tamanho das tabelas de encaminhamento, o que pode ser feito com base nas soluções abaixo descritas.

Editores Anunciados Consiste em manter em cada nó, de forma semelhante às subscrições, informação de anúncios de publicação associados a cada canal. Ou seja, nós que actuam como editores anunciam-se como tal. O objectivo é poder determinar com base na comparação das subscrições e anúncios de publicação associados a cada canal se vão ser geradas notificações por esse canal. Desta forma evita-se a propagação de subscrições a locais da rede onde não existem nós a gerar notificações relevantes e, consequentemente, diminui-se o tamanho das tabelas de encaminhamento.

O benefício desta aproximação tanto maior quanto menor for o conjunto de subscrições intersectado com o conjunto de anúncios de publicação. Em particular, significa que existem poucos nós a gerar notificações para um dado conjunto de subscrições.

Esta método é usado por exemplo no sistema Siena [3].

Compressão de Tabelas de Subscrições A compressão de tabelas de subscrições pode ser feita através da agregação de entradas das mesmas. Caso uma entrada esteja completamente contida numa segunda (associadas ao mesmo canal) é possível manter apenas a segunda. Este método é preciso uma vez que permite reduzir a tabela sem originar falsos positivos. No caso de admitirmos a existência de falsos positivos podemos substituir as entradas de subscrições S_1 e S_2 por uma única entrada associada a uma subscrição S_3, tal que a união de S_1 e S_2 está completamente contida em S_3. No entanto, podem existir

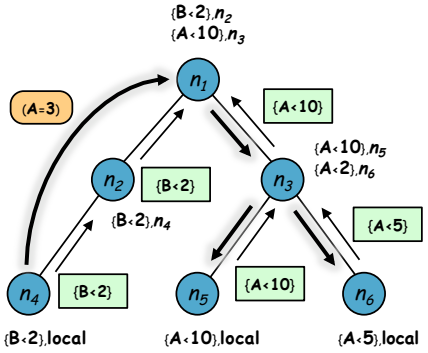

Figura 2.12: Envio de notificações com base em árvores. [20]

outras subscrições compatíveis com S_3, o que resulta em falsos positivos.

O uso de compressão imprecisa permite reduzir fortemente as tabelas e assim, reduzir os custos de computação no processo de encaminhamento. No entanto, se usada de forma demasiado agressiva pode gerar entradas que são compatíveis com todas as subscrições o que faz com que o encaminhamento degenere para uma inundação.

De notar que a compressão das tabelas de anúncios de publicação não se reveste da mesma utilidade face à complexidade de implementação, uma vez que estes são apenas testados durante a propagação de subscrições.

O sistema XNet [8], uma rede endereçada em termos de conteúdo XML, usa tabelas de encaminhamento e implementa algoritmos de compressão das mesmas através de agregação baseada em factorizações de árvores de subscrições [9].

Árvores de Disseminação Árvores de disseminação são outra possível aproximação para reduzir o tamanho das tabelas de encaminhamento. Basicamente assenta no princípio de estruturar a rede de forma a limitar de forma hierárquica os vizinhos dos nós. Esta estruturação é feita com base nos princípios das redes sobrepostas.

Perante uma estrutura em árvore um nó só precisa de manter informação sobre a sua subárvore e, como tal, o tamanho das tabelas decresce em função do nível da árvore. A eficiência desta redução depende do balanceamento da árvore.

Perante esta estrutura, as notificações podem ser enviadas directamente para a raiz e depois propagadas a partir da raiz pelos canais compatíveis ou podem ser enviadas para cima em direcção à raiz passando pelos nós intermédios que por sua vez podem propagar as notificações para baixo pelos canais apropriados.

O sistema Hermes [23], é um exemplo desta aproximação e usa a rede sobreposta Pastry [28] para criar as árvores de disseminação.

O *broadcast* de subscrições necessário para o preenchimento e manutenção das tabelas

de encaminhamento gera duplicados e por consequência um desperdício de largura de banda. De forma a evitar esta situação podem ser criadas previamente árvores (*spanning tree*).

Uma forma de criar estas árvores é usar redes sobrepostas. Uma aproximação, denominada *mesh-first*, consiste em usar redes sobrepostas estruturadas genericamente em grafos e usar os mecanismos de comunicação uni-ponto oferecidos pelas mesmas em conjunto com algoritmos semelhantes aos usados nos protocolos de *multicast* IP.

Por oposição à anterior, uma solução *tree-first*, consiste em estruturar logo à partida a rede sobreposta como uma árvore. Neste caso é necessária especial atenção para lidar com eventuais ciclos durante reorganizações da árvore.

A criação de árvores e a manutenção de tabelas de subscrições, são características desta aproximação que acarretam problemas relacionados com a entrada e saída de participantes do sistema. Se o sistema apresentar elevado dinamismo, existe um custo associado a toda a reconfiguração que tem de ser feita em concordância. Durante estas reconfigurações mensagens podem ser perdidas pelo que é necessário que exista um protocolo de recuperação dessas falhas.

2.3.3 Mapeamento de Notificações e Subscrições em Espaços de Chaves

Como referido *Content Based Routing* pode ser feito recorrendo a redes sobrepostas. *Distributed Hash Tables*, um tipo particular de rede sobrepostas, provaram deter propriedades de escalabilidade, encaminhamento e robustez bastante interessantes para um paradigma editor/assinante descentralizado. Numa *dht* nós e objectos são mapeados para espaços de chaves. Uma possível solução para realizar *Content Based Routing* consiste em mapear notificações e subscrições em espaços de chaves e usar este tipo de redes sobrepostas e tirar partido da relação que existe entre o mapeamento e o encaminhamento neste tipo de redes.

Num sistema deste tipo, o registo de um nó X é feito junto dos nós responsáveis pelas chaves para quais a subscrição de X se mapeia. De forma análoga, o encaminhamento de uma notificação é feito para os nós responsáveis pelas chaves para quais a notificação se mapeia.

O sistema EDN [34] foi uma das primeiras abordagens a este tipo de mapeamento em chaves. Para cada evento e subscrição o sistema gerava uma assinatura baseada nos seus atributos que era por sua vez alvo de *hashing* de acordo com a rede sobreposta.

Idealmente o encaminhamento de uma notificação deve usar apenas os nós interessados nessa mesma notificação. No entanto, caso a rede sobreposta não seja estruturada semanticamente para um esquema de *Content Based Routing* é inevitável gerar falsos positivos no que toca a recepção de notificações.

Uma desvantagem desta aproximação, e relacionada também com o parágrafo anterior, consiste em que o mapeamento e os algoritmos dependem da rede sobreposta e por isso não são independentes do esquema usado. As propriedades de encaminhamento

e tolerância a falhas também são igualmente particulares de cada rede usada. Tal dependência do esquema implica que seja feito estudo pormenorizado dos objectivos do sistema de forma a medir as vantagens e desvantagens de uma abordagem específica.

2.4 O Sistema Livefeeds

O sistema Livefeeds, alvo desta dissertação, consiste numa abordagem ao problema da disseminação filtrada associado ao paradigma editor/assinante. O sistema Livefeeds não segue estritamente nenhuma das abordagens anteriormente descritas mas reveste-se de um modelo em que várias características dessas abordagens se encontram presentes. Usando uma arquitectura *peer-to-peer*, o Livefeeds faz uso de uma rede sobreposta e das suas propriedades de organização mas não existe um mapeamento das subscrições e notificações para o espaço de chaves.

As *dht's* de primeira geração como, *chord* e *pastry* depressa evidenciaram e fizeram padrão dos conceitos associados a um sistema *peer-to-peer*. À semelhança do que acontece nos sistemas *peer-to-peer* descritos, o sistema Livefeeds é completamente descentralizado e não existe qualquer tipo de sincronização directa entre os participantes. A tolerância a falhas presente em todos os sistemas apresentados é também um aspecto que damos grande importância no Livefeeds.

Em termos de organização, encontramos algumas semelhanças com o sistema descrito na secção 2.2.3.7, com a noção de partição também existente no Livefeeds e o estudo de novas abordagens com visibilidade completa. Na mesma secção, discutimos também que os autores estudam também soluções de visibilidade parcial à semelhança do que propomos nesta dissertação. O compromisso entre o número de nós que são afectados pela chegada do novo participante e a taxa de novos nós é algo que é premente na discussão de todos os sistemas *peer-to-peer*.

Nas secções seguintes descrevemos o estado actual do sistema Livefeeds e discutimos algumas das limitações que motivam esta dissertação.

2.4.1 Caracterização de um Nó Participante

No sistema Livefeeds, cada nó é definido pelo seu identificador único ou chave, o seu endereço e o seu filtro. O filtro de um nó representa um padrão de eventos que este pretende vir a receber. Cada participante do sistema mantém uma base de dados que representa o seu conhecimento do sistema. Este conhecimento consiste nas chaves, endereços e filtros dos restantes participantes. Dizemos assim que cada nó possui uma vista completa do sistema. Dos vários atributos (chave, endereço e filtro) que caracterizam um nó, o filtro é aquele que assume potencialmente uma maior dimensão.

Os nós participantes podem ser editores, assinantes ou ambos. No entanto, todos os nós participantes podem ser, durante a sua vida, chamados a efectuar trabalho nas disseminações de eventos.

O espaço de chaves encontra-se dividido em partições ou fatias sendo que o nó presente em cada fatia com o menor identificador é considerado o *sequenciador* e assume um papel especial no funcionamento do sistema descrito abaixo. De notar que não existe sincronização entre os nós sobre a decisão de quem é o sequenciador. Cada nó determina o seu sequenciador a partir do conhecimento que detém na altura, o que pode causar momentâneamente a existência de mais do que um sequenciador por partição. Ainda assim, à medida que a visão de cada nó sobre o sistema converge haverá um maior acordo entre os participantes em relação a que nós são sequenciadores.

2.4.2 Algoritmo de Filiação

O sistema Livefeeds é um sistema dinâmico, ao longo do tempo novos nós chegam ao sistema e os nós presentes do sistema devem ficar a conhecer estes novos nós. Por outro lado, existem nós que abandonam o sistema, quer por falha ou vontade própria. A saída destes deve ser do conhecimento dos restantes participantes para que estes possam proceder à limpeza das suas bases de dados. O conhecimento da chegada e saídas de nós é produzido pelo algoritmo de filiação. Este algoritmo é o principal responsável por criar em todos os nós uma visão completa dos participantes do sistema, disseminando as entradas e saídas dos novos nós pelos sistema.

2.4.2.1 Entrada de um Nó

À chegada ao sistema vamos supor que o novo nó conhece um outro nó que já se encontra no sistema. O nó começa por contactar este nó semente e através deste, obtém a base de dados dos participantes do sistema em termos de identificadores, endereços e chaves dos filtros. Uma vez obtida a base de dados o nó consegue assim determinar o sequenciador da sua fatia e enviar para este o pedido de entrada no sistema.

O sequenciador irá agregar os pedidos durante um período de tempo pré-configurado e iniciar a disseminação das novas entradas. Esta agregação visa minimizar o impacto dos cabeçalhos das mensagens que seria extremamente elevado se fosse iniciada uma disseminação por cada nova entrada no sistema. Durante a disseminação, o conjunto de novos nós, os seus identificadores, endereços e filtros é disseminado entre os nós. Quando o nó recebe notificação da sua própria entrada durante a disseminação este considera que entrou no sistema.

Nesta altura o nó necessita também de ficar a conhecer os filtros dos nós que já se encontram no sistema. Como os filtros são potencialmente os objectos de maior dimensão no sistema, o *download* da base de dados dos filtros é feito, não apenas a partir de um nó mas, a partir de vários de modo a distribuir o *upload* por esses vários nós. Para este efeito, o nó contacta aleatoriamente um nó para que este lhe envie uma parte da base de dados de filtros até obter todas as partes.

2.4.2.2 Algoritmo de Disseminação de Entradas

O sequenciador é responsável por iniciar a disseminação dos pedidos de entrada que agregou. Esta disseminação poderia ser feita colocando o sequenciador a notificar todos os nós do sistema directamente mas essa seria uma abordagem que implicaria que os nós sequenciadores realizariam muito mais trabalho que os restantes nós. Em alternativa, queremos tirar partido da rede estruturada formada pelos participantes do sistema e distribuir ao máximo a carga da disseminação das novas entradas pelos participantes no sistema.

Inicialmente consideramos que o sequenciador tem à sua responsabilidade um intervalo no espaço de chaves. Isto é, um intervalo em que, no final da disseminação, todos os nós devem estar notificados das novas entradas. Este intervalo é inicialmente igual ao intervalo de chaves global do sistema, uma vez que todos os participantes do sistema devem ficar a conhecer os novos nós. O sequenciador pode então dividir este intervalo em G subintervalos e escolher um nó em cada um desses para receber a lista de novas entradas e o subintervalo que ele deverá tratar. Uma vez que o nó escolhido em cada subintervalo é o primeiro desse intervalo, quando os nós do segundo nível vão tratar o seu intervalo podem assumir que os nós que estão para trás da sua chave já foram notificados e por isso o intervalo pode avançar até à sua própria chave. Isto significa que um intervalo recebido da forma $[min, max]$ pode tomar a forma $[k, max]$ onde k é a chave do nó. Os nós do segundo nível repetem o processo de divisão do intervalo de forma análoga ao realizado inicialmente pelo sequenciador e assim sucessivamente nos níveis seguintes até que todos os nós foram notificados da chegada dos novos nós. Este processo de subdivisão dos intervalos dá origem a uma árvore de disseminação. O grau G desta árvore pode ser controlado parametrizando o número de subintervalos que devem ser gerados por cada nó a cada nível. A certa altura, os intervalos serão tão pequenos que os nós podem decidir contactar todos os nós do intervalo directamente terminando assim a subdivisão.

Uma versão resumida do algoritmo em pseudo-código pode ser consultada abaixo.

Algoritmo 1 TRATAREVENTOENTRADAS(s, m)

Require: Evento m, Socket s.

```
1: GUARDARNOVASENTRADAS(m.entradas)
2: m.intervalo ← AVANCARINTERVALO(m.intervalo, k) {k, a chave do no}
3: if m.intervalo.nodeCount() > G then
4:     {G, grau da árvore de disseminação}
5:     subintervalos ← DIVIDIRINTERVALO(m.intervalo, G)
6:     for all intervalo : subintervalos do
7:         for all n : NÓSCONHECIDOS(intervalo) do
8:             repeat
9:                 resultado ← ENVIAR(NÓSCONHECIDOS(intervalo).next(), m)
10:            until resultado = true ∨ NÓSCONHECIDOS(intervalo).next() = null
11:        end for
12:    end for
13: else
14:    for all n : known(m.intervalo) do
15:        ENVIAR(n, m)
16:    end for
17: end if
```

O algoritmo descrito resulta então numa distribuição do trabalho da disseminação. No entanto, os nós escolhidos para realizar trabalho durante a distribuição seriam sempre os mesmos uma vez que o intervalo seria sempre o mesmo ($[0, chavemax]$). Para melhor balancear a carga entre os vários nós consideramos um intervalo circular e um ajuste segundo um deslocamento aleatório no início de cada disseminação. Podemos então definir o intervalo com deslocamento aleatório, para um espaço de chaves $[0, 1]$ como,

$$[\delta, 1] \vee [0, \delta] \tag{2.1}$$

2.4.2.3 Saída de um Nó

As entradas de novos nós no sistema são um acontecimento que consideramos importante pois tratam-se de novos nós que podem realizar trabalho, as saídas, por outro lado, podem ser consideradas menos importantes uma vez que um nó pode simplesmente passar por cima do nó que falhou. Ainda assim, a acumulação de nós que falharam resulta num desperdício de espaço e em complexidade desnecessária durante as disseminações. Para além do desperdício de espaço em cada nó a falta de limpeza das bases de dados dos nós fará com que o custo do *download* da base dados inicial aquando da entrada do nó seja muito mais despendioso. Por essas razões, queremos acelerar o processo em que os nós tomam conhecimento da saída dos nós.

Quando um nó detecta que é incapaz de comunicar com outro nó do sistema considera que este abandonou o sistema. Nessa altura o nó pode enviar uma notificação para

o sequenciador ou iniciar ele próprio uma disseminação notificando os restantes participantes da saída do nó.

Uma vez que os participantes do sistemas comunicam entre si de forma concorrente, isto significa que a detecção de saídas é concorrente. Este facto revela-se particularmente problemático quando existe um aumento repentino da taxa de saídas, por exemplo, devido a uma partição na rede. A saída de um nó pode ser detectada por vários nós concorrentemente gerando um elevado tráfego de notificações ao sequenciador. Por outro lado se os nós iniciarem a sua própria disseminação iremos ter várias disseminações redundantes.

Numa primeira versão do Livefeeds os nós limitam-se a iniciar uma disseminação do evento de saída o que acarreta o problema de existirem várias disseminações redundantes e um consumo exagerado de largura de banda. Na secção 2.4.4 descrevemos uma variação do sistema que pretende lidar com este problema.

2.4.3 Recuperação de Falhas

Os algoritmos acima descritos baseiam-se num sistema de comunicação subjacente que é essencialmente *best effort*. Isto significa que não existe garantia de que as mensagens são entregues durante a disseminação e como tal podem ocorrer falhas.

2.4.3.1 Omissões durante a Disseminação de Entradas

As omissões ao nível da disseminação das novas entradas significam que nós poderão não receber a informação da chegada de novos nós. Deste modo o nó não poderá contactar os novos nós durante as disseminações seguintes. Estas omissões resultam da falha de nós durante as disseminações ou da existência de disseminações concorrentes.

Para recuperar a informação de eventos passados, os sequenciadores identificam cada evento da agregação de novas entradas/saídas com uma estampilha única. Posteriormente, os nós escolhem aleatoriamente um outro nó que conhecem no sistema e comparam entre si as vistas de estampilhas conhecidas. Deste modo, cada um fica a saber que estampilhas tem em falta e podem trocar os eventos em falta entre si. Este processo, repetido ao longo do tempo em cada nó no sistema, consiste num mecanismo global de carácter epidémico que irá levar os nós a convergirem para o mesmo conhecimento de eventos passados.

Uma vez que o processo de escolha de nós é aleatório é impossível garantir formalmente que o sistema é tolerante a falhas uma vez que seria sempre possível uma sequência de reparações em que as falhas existentes fossem impossíveis de reparar. No entanto, podemos dizer que o sistema é tolerante a falhas com alta probabilidade como foi demonstrado em [12].

2.4.4 Lidando com picos de Churn

Acima descrevemos o processo de divulgação de novas entradas e saída de nós do sistema. O processo descrito com base no uso de nós agregadores (sequenciador) para as entradas e, as disseminações iniciadas pelos nós quando detectada a saída de um nó, sofrem no entanto quando existe um aumento extraordinário da taxa de entrada ou saídas. Uma subida repentina da taxa de entradas poderá levar a sobrecarga dos sequenciadores enquanto o súbito aumento da taxa de saídas dá origem a diversas disseminações redundantes que resultam num desperdício de largura de banda.

Para lidar com este problema foi proposta a introdução de uma alteração ao livefeeds descrito acima, que tem como base a noção de um árvore de agregação. Na solução descrita acima, podemos dizer que existe um nível de agregação. Os pedidos de entrada são enviados para os sequenciadores e esses mesmos iniciam a divulgação dos novos nós.

Consideremos, em variação, que os pedidos de entrada são recebidos por um sequenciador mas, sendo definido um alvo para a taxa de agregação de entradas feita no nó, este pode decidir encaminhar o pedido para outro sequenciador caso a taxa de agregação esteja aquém do alvo, dando origem a mais um nível na agregação. Iremos obter uma árvore de agregação de altura h em que no nível h existirão um maior número de sequenciadores que no nível $h-1$ e assim sucessivamente. Quando a taxa de entradas é muito elevada os sequenciadores nos níveis mais baixos iram atingir mais rapidamente o alvo de agregação e por isso as disseminações das entradas serão iniciadas nos níveis mais baixos aliviando os sequenciadores dos níveis superiores. Por outro lado, quando a taxa de entradas é menor mais facilmente os pedidos chegam aos níveis superiores. Para evitar um aumento considerável da latência associada à entrada/saída de um nó possibilita-se que, quando a taxa de entradas seja reduzida, a àrvore de agregação seja truncada sendo o pedido enviado para a raiz saltando os níveis de sequenciadores intermédios.

Um processo semelhante ocorre para a divulgação das saídas. Quando um nó detecta a saída de outro, o primeiro não inicia logo uma disseminação. O nó determina um nó sequenciador do nível mais baixo e envia para este notificação da saída do nó.

Ao introduzir sequenciadores para as saídas reduzimos a probabilidade de ocorrerem disseminações redundantes e o uso de uma árvore de agregação permite acomodar os picos nas referidas taxas de entrada e de saída de participantes.

2.4.5 Custo do Algoritmo de Filiação

A principal limitação do sistema é o custo associado ao algoritmo de filiação com visão total. O custo deste algoritmo será função do dinamismo do sistema, por vezes denominado *churn*. Quanto mais entradas e saídas ocorrem no sistema mais alterações na filiação devem ser propagadas de forma a estas reflectirem-se em todos os nós.

Para avaliar o custo do algoritmo um modelo de *churn* é usado. Este modelo caracteriza as chegadas de novos nós e o seu tempo de permanência no sistema. O algoritmo de filiação do Livefeeds foi avaliado em função de um comportamento de utilizadores

que se pensa ser semelhante aos utilizadores do sistema Skype [15]. Isto porque o uso do sistema Skype é maioritariamente diurno e análogo ao uso da *Web* por oposição a sistemas *peer-to-peer* usados para partilha de ficheiros. O sistema Livefeeds tratando-se de um sistema de acesso a conteúdo, apresenta um comportamento também análogo ao uso da *Web*.

Estudo do algoritmo Livefeeds [22] revela então que, considerando a disseminação das novas entradas uma árvore de altura h e em que os nós interiores possuem grau G, é possível aproximar a largura de banda usada em *upload* e *download* como,

$$b_u = b_d = m \cdot r \tag{2.2}$$

O custo do algoritmo de filiação é função da entrada de novos nós, r, e o seu valor principalmente dependente do tamanho dos filtros dos nós. Isto é, o valor de m diz respeito principalmente ao tamanho do filtro. Para um filtro de 500 $bytes$ e uma taxa de chegada de $1.6ns/s$ os nós usariam uma capacidade de largura de banda de *downstream* e *upstream* de $r \times m = 1.6 \times 500 = 800$ bytes/s. Ou seja, uma largura de banda de 6.4 Kbps. Isto implica que quanto maior for a taxa de chegada de nós maior será a largura de banda necessária.

2.4.6 Algoritmo de Disseminação Filtrada

Criada a visão total do sistema nos participantes esta é usada como suporte ao processo da difusão filtrada. Quando um nó editor gera um evento, o nó dá início a uma disseminação semelhante à iniciada pelo sequenciador para disseminar as entradas e saídas. Os nós recebem da mesma forma intervalos que devem tratar esses intervalos são por sua vez divididos em subintervalos e percorridos sequencialmente até encontrar o primeiro nó cujo filtro aceita o evento, sendo o trabalho delegado a esse nó.

Quando um nó recebe um intervalo, uma vez que os nós conhecem todos os filtros, é seguro assumir que os nós cuja chave é menor que a chave do nó já foram considerados e por isso o intervalo pode avançar até à chave do nó tomando a forma $[k, max]$. Através da subdivisão e o referido avanço os intervalos diminuem até que os nós podem ser contactados directamente terminando assim a disseminação.

Uma versão resumida do algoritmo em pseudo-código pode ser consultada abaixo.

Algoritmo 2 TRATAREVENTODISSEMINACAOFILTRADA(s, m)

Require: Evento m, Socket s.

```
1:  m.intervalo ← AVANCARINTERVALO(m.intervalo, k) {k, a chave do no}
2:  if m.intervalo.NodeCount() > G then
3:      {G, grau da árvore de disseminação}
4:      subintervalos ← DIVIDIRINTERVALO(m.intervalo, G)
5:      for all intervalo : subintervalos do
6:          for all n : NÓSCONHECIDOS(intervalo) do
7:              repeat
8:                  if n.filtro.aceita(m) then
9:                      resultado ← ENVIAR(NÓSCONHECIDOS(intervalo).next(), m)
10:                 else
11:                     {avançar...}
12:                 end if
13:             until resultado = true ∨ NÓSCONHECIDOS(intervalo).next() = null
14:         end for
15:     end for
16: else
17:     for all n : known(m.intervalo) do
18:         ENVIAR(n, m)
19:     end for
20: end if
```

Durante a disseminação apenas são seleccionados nós cujo filtro aceita o evento não são gerados falsos positivos e a árvore aleatória gerada permite um balanceamento da carga da disseminação. Adicionalmente, podemos garantir que não são gerados falsos negativos na ausência de falhas caso os nós assegurem que entregam a mensagem a um nó cuja vista contenha a sua. Isto é, o nó a que é entregue a mensagem não pode possuir uma visão menor do sistema pois isso levaria à possibilidade de nós que aceitariam o evento e que não façam parte da sua vista nunca mais serem contactados durante essa disseminação. Para este efeito, os nós enviam juntamente com o evento a sua vista para que o nó ao receber o evento possa comparar as vistas e decidir se aceita ou não continuar a disseminação.

2.4.7 Custo do Algoritmo de Disseminação Filtrada

O custo do algoritmo de disseminação filtrada representa a largura de banda despendida nas disseminações de eventos que são publicados no sistema. Prever este valor à partida é díficil na medida em que a largura de banda despendida por um nó depende fortemente do seu filtro. Se o nó possuir um filtro muito abrangente, este vai participar num maior número de disseminações. Por outro lado, se o filtro do nó for muito específico, menos largura de banda este usará, uma vez que participa em menos disseminações.

Na secção 2.4.5 concluímos que o custo médio de disseminações consecutivas de

eventos que ocorrem a uma taxa r seria igual a $m \cdot r$, onde m é o tamanho da mensagem associada ao evento.

Uma rude estimativa do custo do algoritmo da disseminação filtrada pode ser obtida estimando a percentagem média dos eventos que os nós aceitam face ao total dos eventos que ocorrem. Por exemplo, se em média os nós aceitarem p % dos eventos que ocorrem significa que a taxa de novos eventos é para estes nós igual a $p \cdot r$ e a largura de banda despendida em média pelos nós igual a $p \cdot r \cdot m$.

Em conclusão, uma vez que o custo depende da natureza das mensagens disseminadas, da taxa de ocorrência de eventos e dos filtros dos nós. O custo da disseminação é extremamente variável. Se o sistema se limitiar a enviar a disseminar notificações entre os participantes o tamanho das mensagens será relativamente pequeno. Por outro lado, se disseminarmos o objecto do próprio evento podemos ter um custo muito mais elevado.

Desta forma, o custo da disseminação é uma característica da aplicação que usa o sistema Livefeeds. Diferentes aplicações terão utilizadores com diferentes comportamentos variando assim os vários factores que influenciam o custo da disseminação.

2.4.8 Limitações da versão Livefeeds actual

Apesar de diversas optimizações o algoritmo de filiação com visão completa do sistema é relativamente caro, um custo mensurável em termos de largura de banda. Possuir uma visão completa de um sistema altamente dinâmico limita assim a escalabilidade do mesmo uma vez que existe um grande número de novas entradas e saídas para disseminar.

A entrada de novos nós significa que novos filtros tem de ser dados a conhecer a todos os nós fazendo com que o problema do dinamismo do sistema seja agravado pela complexidade dos filtros. Caso os filtros possuam um tamanho elevado, a sua propagação por todos os participantes terá custos, em termos de largura de banda, mais elevados em função desse mesmo tamanho.

Concluímos também anteriormente que as disseminações filtradas terão um custo que depende da aplicação que usa o livefeeds como base. Aplicações que disseminam eventos com elevado tamanho, produzem eventos muito frequentemente e tem utilizadores com filtros muito abrangentes vão despender uma maior largura de banda.

De forma a permitir um maior leque possível de aplicações é importante que o custo dos algoritmos do livefeeds seja o mais reduzido possível. Por isso, é de todo o interesse explorar alternativas. Uma dessas alternativa encontra-se numa solução com visão parcial em que, os nós deixam de conhecer a totalidade dos nós do sistema mas apenas parte deste. Essa redução permite a diminuição da largura de banda necessária permitindo que o sistema escale de forma aproximadamente proporcional em relação à redução do estado mantido. Por outro lado, a redução da visão que cada participante possui do sistema introduz potencialmente dificuldades durante o algoritmo de disseminação filtrada. Isto é, não conhecendo o filtro de todos os nós não podemos garantir que a mensagem não

é entregue a um nó que não a aceite, gerando um falso positivo. No capítulo seguinte descrevemos a solução e proposta e procemos à análise do compromisso referido e de possíveis estratégias para mitigar os problemas que esta acarreta.

3

Concepção e Análise da Solução

Na secção 2.4 foi feita uma descrição do sistema Livefeeds. Referimos que cada participante conhece as chaves, endereços e filtros de todos os participantes do sistema e consideramos que os nós possuem uma visão completa do sistema. Concluímos também que manter essa visão em sistemas que sejam alvo dum grande dinamismo em termos de entradas e saídas é uma limitação dessa mesma abordagem.

A solução que pretendemos descrever e analisar nesta dissertação passa por reduzir a visão que cada nó tem do sistema. Deste modo, pretendemos reduzir os custos associados à actualização das visões dos participantes que resultam dos eventos de chegada e saída de nós.

A largura de banda despendida na manutenção da visão completa diz respeito em grande medida aos filtros dos novos nós que tem de ser dados a conhecer aos participantes que já se encontram no sistema. Por isso, decidimos numa primeira abordagem limitar a visão que os nós possuem do sistema em termos dos filtros. Ou seja, os nós continuam a ter uma visão completa do sistema em termos de chaves e endereços mas irão apenas conhecer uma parte dos filtros do sistema.

3.1 Caracterização de um Nó Participante

Nesta abordagem cada nó é igualmente definido pela sua chave, o seu endereço e o seu filtro. Durante a sua existência um nó manterá uma base de dados onde guarda as chaves e endereços de todos os participantes do sistema que fica a conhecer bem como um *hash* dos seus filtros, a que podemos chamar a chave do filtro. Este conhecimento das chaves dos filtros será útil durante a disseminação filtrada e a sua vantagem explicada mais à frente. Adicionalmente, o nó mantém uma base de dados de filtros que, ao contrário do

que acontecia na aproximação de visibilidade completa, vai ser limitada a um intervalo em função da chave do nó. Criamos assim a noção de *vizinhança* do nó definida por um dado parâmetro d. O intervalo que representa a vizinhança dum nó toma a forma de $[k-d, k+d]$ sendo k a chave do nó.

3.2 Alterações ao Algoritmo de Filiação

Com a criação da noção de vizinhança no que diz respeito à visão dos filtros dos restantes participantes o algoritmo de filiação requere algumas alterações. A necessidade destas mudanças, resulta do facto de que o conhecimento que cada nó deve ter após a entrada de um nó pode diferir, por oposição à versão de visibilidade completa em que o conhecimento seria igual em todos os nós. Existem nós que devem ficar a conhecer o filtro e outros que não. Como tal, torna-se demasiado complexo produzir esse diferente conhecimento numa só disseminação. Em vez disso, optamos por manter uma disseminação global que dissemina as chaves, endereços e chaves dos filtros dos novos nós e uma outra disseminação que diz respeito apenas aos filtros.

3.2.1 Entrada de um Nó

À chegada ao sistema os nós exibem um comportamento em tudo semelhante à versão anterior. Fazem o *download* base de dados dos participantes do sistema em termos de identificadores, endereços e chaves dos filtros através de um nó semente. De seguida, contactam o sequenciador e enviam para este o pedido de entrada no sistema. O sequenciador agrega os pedidos e inicia uma disseminação que desta vez não vai conter os filtros. Esta disseminação segue o algoritmo de disseminação de entradas/saídas descrito na secção 2.4.2.2.

3.2.2 Descarregamento da Base de Dados Inicial de Filtros

Na versão anterior, o nó obtinha a base de dados de filtros a partir de qualquer nó. Isto era possível uma vez que todos os nós tinham um visão completa do sistema. Na versão de visibilidade parcial os nós apenas conhecem os filtros dos seus vizinhos e por isso, à chegada, um novo nó apenas pode obter os filtros que deve conhecer a partir de nós que partilham a sua base de dados. Uma vez que a vizinhança é definida em relação a cada chave não existe um nó único no sistema que pode ser contactado para obter a totalidade do conhecimento de filtros. No entanto, por definição, os intervalos de vizinhança dos nós intersectam-se como demonstrado no exemplo da figura 3.1.

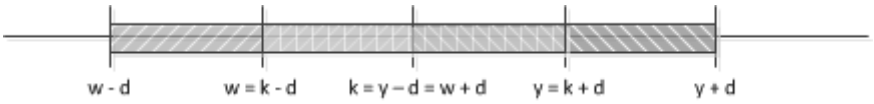

Figura 3.1: Exemplo de intersecção de vizinhança de nós

Isto significa que os filtros no intervalo $[k, k + d]$ podem ser obtidos a partir de qualquer nó desse intervalo, enquanto que, os filtros no intervalo $[k - d, k]$ podem ser obtidos através dum nó desse intervalo. Dado que o *download* da base de dados de filtros ocorre num momento em que o nó já terá feito o *download* dos endereços, o nó pode facilmente contactar nós nesses intervalos específicos. Adicionalmente, tendo em conta que os filtros são potencialmente os objectos de maior dimensão no sistema o *upload* dos filtros é divido entre vários nós. O nó que pretende fazer o *download* divide a sua vizinhança em vários intervalos contactando para cada um desses um nó existente no sistema cuja vizinhança contém esse intervalo.

3.2.3 Disseminação de Filtros

Uma das principais diferenças entre o novo algoritmo de filiação e o presente na anterior abordagem do Livefeeds consiste na forma como os nós ficam a conhecer os filtros dos restantes participantes. Na versão anterior, o conhecimento dos filtros era obtido durante a disseminação de entradas. Enquanto que previamente todos os nós deveriam ficar a conhecer os filtros dos novos nós, agora, os nós só devem receber os filtros dos nós na sua vizinhança. Isto significa que os filtros que os nós devem ficar a conhecer são diferentes para cada nó. Perante este facto, faz sentido separar a disseminação de filtros da disseminação de entradas.

Conforme descrito anteriormente, o nó considera que entrou no sistema quando recebe notificação da sua entrada durante a disseminação de entradas. Nesta altura, o próprio nó irá iniciar a disseminação do seu filtro que deverá ter como alvo todos os nós cuja vizinhança contém o nó. Uma vez que o intervalo que representa a vizinhança dos nós tem a mesma largura ($2d$) para todos os nós do sistema, a chave do nó (k) pertencerá à vizinhança de todos os nós que se encontram à distância d, isto é, o intervalo $[k-d, k+d]$ (a própria vizinhança do nó).

A disseminação do filtro é feita de forma semelhante ao algoritmo de disseminação de entradas, distribuindo o trabalho da disseminação pelos nós presentes no intervalo. O algoritmo encontra-se descrito em pseudo-código mais abaixo (Algoritmo 3).

Algoritmo 3 TRATAREVENTOFILTRO(s, m)

Require: Evento m, Socket s.

```
1: GUARDARFILTRO(m.filtro)
2: m.intervalo ← AVANCARINTERVALO(m.intervalo, k) {k, a chave do no}
3: if m.intervalo.nodeCount() > G then
4:     {G, grau da árvore de disseminação}
5:     subintervalos ← DIVIDIRINTERVALO(m.intervalo, G)
6:     for all intervalo : subintervalos do
7:         for all n : NÓSCONHECIDOS(intervalo) do
8:             repeat
9:                 resultado ← ENVIAR(NÓSCONHECIDOS(intervalo).next(), m)
10:            until resultado = true ∨ NÓSCONHECIDOS(intervalo).next() = null
11:        end for
12:    end for
13: else
14:    for all n : known(m.intervalo) do
15:        ENVIAR(n, m)
16:    end for
17: end if
```

3.3 Recuperação de Falhas

O algoritmo acima descritos baseiam-se num sistema de comunicação subjacente que é essencialmente *best effort*. Isto significa que não existe garantia de que as mensagens são entregues durante a disseminação e como tal podem ocorrer falhas.

3.3.1 Omissões durante a Disseminação de Entradas/saídas

A recuperação de falhas durante a disseminação de entradas/saídas, que contém os endereços, chaves e chaves de filtros, mantém-se inalterada e segue o processo anteriormente descrito na secção 2.4.3.

3.3.2 Omissões durante a Disseminação de Filtros

Ao nível da disseminação de filtros, a perca de mensagens irá resultar em que nós possam ficar sem conhecer filtros que deveriam fazer parte da sua base de dados de conhecimento. A reparação de filtros não pode ser feita com qualquer nó do sistema uma vez que nem todos os nós do sistema possuem o conhecimento dos filtros necessário a cada nó. Mais precisamente, o conhecimento de filtros é diferente para cada nó uma vez que a vizinhança é definida em relação a cada chave. Isto significa que não existe um nó único no sistema que pode ser contactado para comparar a totalidade do conhecimento de filtros.

De forma semelhante ao que se passa no *download* inicial da base de dados de filtro, verifica-se que os filtros no intervalo $[k, k + d]$ podem ser reparados por comparação com

qualquer nó desse intervalo, enquanto que, os filtros no intervalo $[k-d, k]$ podem ser reparados através dum nó desse intervalo.

A reparação dos filtros segue um processo semelhante ao da reparação do conhecimento de eventos de entrada. O nó escolhe aleatoriamente um nó nos intervalos referidos acima e envia para este o conjunto de filtros que determina como estando em falta nesse intervalo. O nó determina este conjunto calculando a diferença entre o conjunto de nós que conhece no intervalo e os filtros que conhece nesse mesmo intervalo. O segundo nó, ao receber o pedido de reparação responde com a intersecção do conjunto de filtros que conhece e o conjunto de filtros em falta recebido. Adicionalmente, o segundo nó envia também para o primeiro nó o seu conjunto de filtros em falta que o primeiro nó tratará da mesma forma.

3.4 Algoritmo de Disseminação

3.4.1 Alterações ao Algoritmo de Disseminação Filtrada

Na versão de visibilidade completa os nós conheciam os filtros de todos os participantes e podiam assim garantir que não entregavam um evento a um nó cujo filtro não aceitasse o evento. Reduzindo a visibilidade do nó é impossível garantir, quando contactado um nó fora da vizinhança, que o nó aceita o evento. Isto é, o nó só pode testar durante a disseminação se um nó aceita um evento se este pertencer à sua vizinhança.

Na versão anterior referimos também que quando um nó recebia um intervalo durante a disseminação este podia assumir que todos os nós do intervalo predecessores à sua chave já tinham sido considerados. No entanto, isso só podia ser assumido quando todos os filtros eram conhecidos. Os nós predecessores podem ter sido contactados por nós que não conheciam os seus filtros e, por isso, gerando potencialmente um falso positivo. Um intervalo da forma $[a, b]$ que ao avançar poderia tomar a forma $[k, b]$, só o pode fazer caso se verifique

$$a \in [(k-d)+1, k]$$

significando que o início do intervalo a tratar está contido no conjunto dos predecessores pertencentes à vizinhança do nó. Eventualmente os intervalos ficam tão pequenos que passam a estar contidos na vizinhança dos nó sendo assim possível testar os filtros dos nós do intervalo. O problema que existe ao não avançar o intervalo prende-se com a possibilidade de gerar duplicados. Para evitar esse acontecimento os nós contactados são registados na mensagem disseminada entre os nós.

Apesar do conhecimento que os nós possuem dos filtros ser parcial, é garantido, na ausência de falhas, que não existem falsos negativos. Uma vez que os intervalos não avançam até à condição referida acima garantido que serão tratados por um nó que conhece os filtros dos nós nesse intervalo. Adicionalmente como referido anteriormente, é necessário também garantir que a vista do nó a que se entrega um evento contém pelo

menos a vista do nó emissor.

3.4.2 Tirando partido da existência de Filtros Redundantes

Num sistema de disseminação filtrada associada ao paradigma editor/assinante em que os nós definem os seus interesses por meio de filtros, existe sempre a possibilidade de existirem nós com filtros idênticos. A probabilidade de existirem filtros redundantes é tanto maior quanto menor for o número de diferentes classes de eventos.

Quanto maior for a redundância de filtros, maior a probabilidade de um filtro pertencente a um nó exterior à vizinhança do nó ser idêntico ao filtro de um nó conhecido. Como referido anteriormente, apesar de não conhecerem os filtros de todos os participantes, os nós conhecem as chaves dos filtros de todos os participantes. Os nós podem assim, no momento de contactar um nó exterior à sua vizinhança, testar a chave do filtro desse nó em relação às chaves dos filtros pertencentes à sua vizinhança. Se a chave do nó exterior pertencer ao conjunto de chaves de filtros pertencentes à vizinhança então significa que o nó conhece também o filtro do nó exterior, podendo assim testar se o filtro que conhece aceita o evento.

3.5 Análise da Solução

Nas secções anteriores apresentámos uma nova versão do sistema Livefeeds. Nesta versão, os participantes deixaram de ter um conhecimento completo dos filtros dos restantes nós do sistema. Esta nova aproximação permitirá reduzir os custos de largura de banda associados ao algoritmo de filiação. Por outro lado, ao possuírem um conhecimento parcial do sistema, os nós terão maior dificuldade em garantir, durante o algoritmo de disseminação filtrada, que não entregam eventos a nós cujo filtro não aceita esse evento. Situação referida também como a ocorrência de falsos positivos.

Nesta secção procuramos analisar com maior detalhe os custos estimados dos algoritmos descritos anteriormente.

3.5.1 Algoritmo de Filiação - Disseminação de Novas Entradas

Na secção 2.4.5 discutimos, de acordo com estudo prévio do sistema Livefeedds [22], que o custo médio, em termos de largura de banda de *download* e *upload*, da disseminação de uma mensagem de tamanho m ao longo duma árvore de grau G seria m. Assim, para uma taxa r de ocorrência de disseminações o custo seria em termos de *download* e *upload* igual a,

$$b_d = b_u = r \cdot m \ (bytes/s) \tag{3.1}$$

Desta forma, concluímos que o custo do algoritmo de filiação de visibilidade completa

podia ser estimado com m representando o peso da entrada de um nó no sistema em termos da sua chave, endereço e filtro.

Na abordagem de visibilidade parcial separámos a disseminação de chaves e endereços da disseminação de filtros uma vez que nem todos os participantes devem ficar a conhecer os filtros. Uma vez que o conhecimento dos filtros encontra-se agora limitado à vizinhança do nó a taxa de disseminações que contém os filtros não é igual à taxa de entrada de nós no sistema. Consideremos v como sendo a fracção do espaço de chaves ($[min_k, max_k], min_k \leq max_k$) a que chamamos de vizinhança do nó calculada através do parâmetro d, que define o intervalo da vizinhança como $[k-d, k+d]$, da seguinte forma,

$$v = \frac{2d}{max_k - min_k}, 2d \leq (max_k - min_k) \tag{3.2}$$

Considerando a chegada de novos nós aleatoriamente distribuída pelo espaço de chaves, podemos dizer que a probabilidade de um novo nó pertencer a vizinhança de um nó participante será,

$$P(k_n \in [k-d, k+d]) = \frac{2d}{max_k - min_k} = v \tag{3.3}$$

Deste modo, a taxa média de disseminações com filtro será de igual a $r \cdot v$ o que resulta, considerando m_k o peso da chaves, filtro, endereços e chave de filtro e m_f o peso do filtro, num custo médio de largura de banda associada à entrada de novos nós,

$$\overline{b_u} = \overline{b_d} = r \cdot m_k + r \cdot v \cdot m_f = r(m_k + m_f v) \tag{3.4}$$

$$\overline{b_u} = \overline{b_d} = r(m_k + m_f v)\ (bytes/s) \tag{3.5}$$

Uma vez que a redução de visibilidade afecta apenas os filtros, a redução da largura de banda despendida não é linear. Isto significa que existe um limite em que não se verificam ganhos práticos em função da redução da visibilidade do nó. Este limite é ainda afectado pelo *overhead* do encaminhamento da mensagem. As mensagem necessitam de ser encaminhadas através de um dos protocolos existentes. Caso a mensagem for encaminhada por TCP/IP, existe para cada mensagem um custo acrescido que diz respeito aos *headers* dos datagramas. Associado ao protocolo TCP temos também o custo do estabelecimento da conexão e o fecho da mesma. Acrescentando o peso associado ao *overhead* TCP, m_h, na expressão 3.5 obtemos,

$$\overline{b_u} = \overline{b_d} = r(m_h + m_k + v(m_h + m_f))\ (bytes/s) \tag{3.6}$$

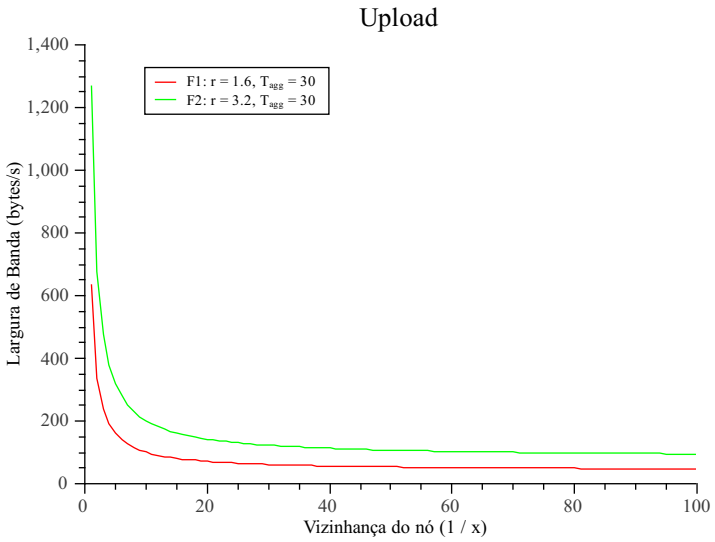

Figura 3.2: Largura de Banda despendida na disseminação de novas entradas

Analisando mais atentamente a expressão 3.6, verificamos que esta modela o comportamento dos sequenciadores como se estes dessem início a uma disseminação por cada pedido que recebem. Este comportamento seria altamente ineficiente uma vez que teríamos a enviar mensagens que acarretam o *overhead* dos respectivos cabeçalhos para disseminar a entrada de apenas um nó quando passado pouco tempo teríamos de repetir o mesmo processo para outro nó.

A solução, já descrita em capítulos anteriores, consiste na introdução de um período de agregação, que irá permitir numa mensagem disseminar a entrada de vários nós reduzindo o impacto dos cabeçalhos. Aumentando o período durante o qual os sequenciadores agregam os pedidos de entrada de novos participantes permite diminuir o *overhead*. No entanto, se este período for muito elevado, teremos um atraso significativo na entrada dos novos nós. Este período de agregação é válido apenas para a disseminação de novas entradas uma vez que a disseminação de filtros é iniciada pelo próprio nó após entrar no sistema. Isto implica que os cabeçalhos tem um impacto elevado na disseminação dos filtros.

Na figura 3.2 verificamos a grande influência do período de agregação no custo de *upload* (o *download* terá o mesmo comportamento). Quanto menor for o tempo entre disseminações menor o número de novas entradas disseminadas em cada evento e por isso o peso dos cabeçalhos em relação a cada entrada é bastante superior.

Actualizando 3.6 obtemos,

$$\overline{b_u} = \overline{b_d} = r(m_k + v(m_h + m_f)) + \frac{m_h}{T_{agg}}\ (bytes/s) \tag{3.7}$$

$$\tag{3.8}$$

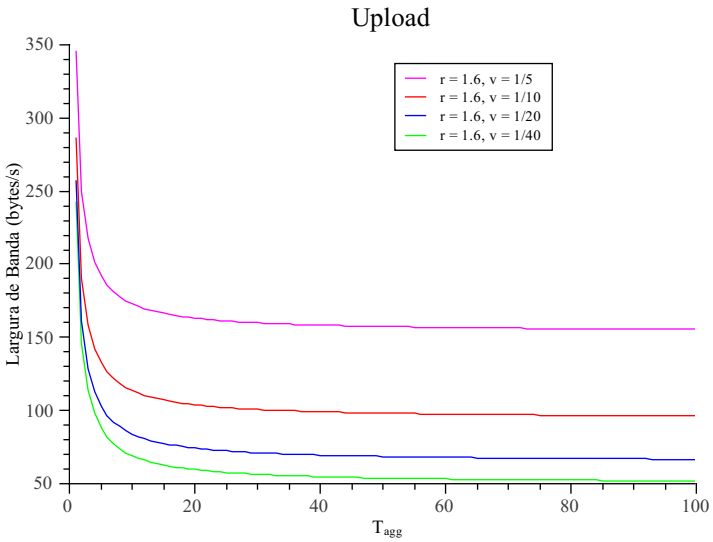

Figura 3.3: Largura de Banda despendida em função da vizinhança do nó

Concluindo que o valor de 30 segundos é adequado para este parâmetro, produzindo uma boa diluição do *overhead*, podemos, na figura 3.3, verificar graficamente a existência do referido limite à redução da largura de banda despendida para duas taxas de chegada de respectivamente 1.6 e 3.2 nós por segundo. O tamanho do filtro usado foi 250 bytes juntamente com 22 bytes que dizem respeito à chave, endereço e chave do filtro do nó e 160 bytes usados como peso associado ao protocolo TCP.

3.5.2 Descarregamento das bases de dados iniciais

A primeira acção que um nó realiza ao chegar ao sistema consiste em, através de um participante existente, obter a base de dados de chaves, endereços e chaves de filtros dos participantes do sistema. Posteriormente, o nó necessita de obter também a base de dados dos filtros dos nós existentes que pertencem à sua vizinhança. Estas duas acções representam um custo significativo nos primeiros momentos de vida de um nó do sistema Livefeeds.

3.5.2.1 Base de Dados de Endereços, Chaves e Chaves de Filtros

No momento do descarregamento da base de dados inicial de endereços esta terá um custo em termos de *download* para o nó relacionado com a dimensão do sistema.

A dimensão do sistema é dada em média pela taxa de chegada de participantes e pela duração média da sua permanência no sistema.

$$N_{avg} = T_{session} \times r \tag{3.9}$$

Uma vez que a base de dados guarda a chave, endereço e chave de filtro, um peso que definimos como m_k, temos que o custo do *dowload* da base dados inicial terá o valor de,

$$d_{bd_k} = T_{session} \times r \times m_k \ (bytes) \tag{3.10}$$

O *download* da base de dados inicial representa um evento único de elevado custo associado à entrada do nó. Quanto menor for a duração da sessão do nó maior o custo relativo desta acção. Por outro lado, quanto maior a duração da sessão do nó, o peso deste *download* inicial é diluído face aos restantes consumos que o nó terá ao longo da sua sessão.

A largura de banda média de *download*, $\overline{b_d}$, despendida no descarregamento da base dados inicial de endereços é então dada em função do tempo médio de sessão dos nós

$$\overline{b_d} = \frac{T_{session} \times r \times m_k}{T_{session}} = r \times m_k \ (bytes/s) \tag{3.11}$$

Em termos de *upload* o nó semente terá de enviar ao novo nó a base de dados inicial pelo que terá um custo, m_{db_k}, igual ao referido acima para o *download*. Considerando que o nó semente é um nó aleatoriamente escolhido entre todos os participantes a probabilidade aproximada de um nó realizar o *upload* referente a uma nova entrada é de,

$$p(seed) = \frac{1}{N_{avg}} \tag{3.12}$$

Desta forma, o *upload* médio associado ao envio da base de dados inicial é estimado em função de cada entrada tendo em conta a probabilidade acima da forma,

$$\overline{b_u} = r \times p(seed) \times T_{session} \times r \times m_k \tag{3.13}$$

$$\overline{b_u} = r \cdot m_k \ (bytes/s) \tag{3.14}$$

Concluímos assim que os custos de *upload* e *download* associados ao descarregamento da base de dados inicial de endereços dependem apenas da taxa de chegada de participantes e do peso associado à chave, endereço e chave de filtro de um nó.

De notar que não contabilizamos o *overhead* dos cabeçalhos uma vez que este é relativamente baixo quando comparado com a dimensão total da base de dados.

3.5.2.2 Base de Dados de Filtros

Posterior ao descarregamento da base de dados inicial de endereços o nó necessita de obter a base de dados de filtros que pertencem à sua vizinhança. Tal como a base de dados de endereços, o custo deste *download* depende da dimensão do sistema e, agora no caso dos filtros, do tamanho da vizinhança do nó. Quanto maior for a vizinhança do nó maior será o número de filtros que o nó deve ficar a conhecer. Desta forma, considerando v como a fracção do intervalo de vizinhança do nó em relação ao espaço global de chaves, o custo associado a este *downnload* toma o valor aproximado de,

$$d_{db_f} = T_{session} \times r \times v \times m_f \ (bytes) \tag{3.15}$$

Verificando-se assim que a largura de banda em termos de *downnload* toma o valor,

$$\overline{b_d} = \frac{T_{session} \times r \times v \times m_f}{T_{session}} = r \times v \times m_f \ (bytes/s) \tag{3.16}$$

Por sua vez, o *upload* que um nó gasta no envio da base de dados de filtros depende da probabilidade com que este é escolhido pelo novo nó. Se consideramos que os participantes são uniformemente distribuídos pelo espaço de chaves então a probabilidade de um novo nó pertencer à vizinhança de um nó escolhido ao acaso tem o valor de v, a fracção da vizinhança do nó. Desta forma, a taxa de chegada de participantes à vizinhança de um dado nó tem o valor de,

$$r_v = r \cdot v \tag{3.17}$$

Cada uma das novas chegadas representa um potencial custo de *upload* dos nós que irão pertencer a vizinhança do novo nó. Tendo em conta que o novo nó reparte esta vizinhança num número n de subintervalos, haverá em cada um destes intervalos um nó que será contactado. A probabilidade de um nó ser contactado depende do número de subintervalos e toma a forma,

$$p(filterseed) = \frac{n}{T_{session} \times r \times v} \tag{3.18}$$

Cada nó terá então de efectuar o *upload* de tantos filtros quantos os nós pertencentes a um subintervalo,

$$u_{db_f} = \frac{T_{session} \times r \times v \times m_f}{n} \ (bytes) \tag{3.19}$$

O custo médio em termos de *upload*, $\overline{b_u}$, em função da chegada de novos participantes toma então o valor de,

$$\overline{b_u} = r_v \times p(filterseed) \times \frac{T_{session} \times r \times v \times m_f}{n} \tag{3.20}$$

$$\overline{b_u} = r \cdot v \cdot m_f \ (bytes/s) \tag{3.21}$$

Confirmamos como seria de esperar a influência do tamanho da vizinhança do nó nos custos associados ao descarregamento da base de dados de filtros. Este é também um importante aspecto que demonstra o benefício da redução da visibilidade dos nós. Ao diminuir o tamanho da base de dados de filtros o tempo que o nó leva a ficar completamente disponível para colaborar nas disseminações filtradas é menor o que representa uma vantagem face à versão anterior.

3.5.3 Saídas

A disseminação de saídas é feita juntamente com a disseminação de entradas. Durante o período de agregação o sequenciador prepara um evento contendo o conjunto de nós que entraram no sistema e o conjunto de nós que abandonaram o mesmo. Para melhor compreensão dos diferentes aspectos inerentes ao processo de filiação, contabilizamos o custo das saídas de forma separada.

O custo da disseminação de saídas dependerá da taxa de saídas. Uma vez que estamos a falar em estimativas de valores médios afirmamos que a partir do valor de tempo médio de sessão dos participantes a taxa de saídas será igual à taxa de entradas.

Para cada saída limitamos-nos a disseminar a chave do nó para simplesmente dar a conhecer aos restantes participantes que o nó já não se encontra no sistema.

Podemos assim descrever o custo em termos de largura de banda associado à disseminação de saídas, sendo m_c o peso em *bytes* da chave do nó, de acordo com a expressão abaixo.

$$b_u = b_d = r \cdot m_c \ (bytes/s) \tag{3.22}$$

3.5.4 Reparação Epidémica

A reparação epidémica, descrita nas secções 2.4.3 e 3.3 consiste num processo que ocorre periodicamente no sistema com o intuito de colmatar falhas que possam ocorrer durante o algoritmo de filiação. O seu custo irá depender principalmente da periodicidade com que este ocorre e da probabilidade de um nó perder eventos de agregação de chegadas/-saídas e eventos de disseminação de filtros.

O mecanismo de reparação divide-se na reparação de endereços e na reparação de filtros.

3.5.4.1 Reparação de Endereços

O mecanismo de reparação de endereços, tem por objectivo permitir aos nós recuperarem eventos da disseminação de novas entradas que possam ter perdido. Uma reparação é um processo pedido-resposta-resposta. Em termos de *upload*, o nó que inicia a reparação terá o custo de enviar o pedido e a última resposta. Em termos de *download* terá de gastar largura de banda com a resposta ao seu pedido inicial. O segundo nó terá os custos de *upload* referente à resposta ao pedido e inicial e o *download* do pedido inicial e da resposta final. Isto significa que para cada pedido de reparação verifica-se aproximadamente,

$$b_u = b_d \tag{3.23}$$

Um pedido de reparação de endereços consiste no envio da vista eventos conhecidos pelo nó. Assim, o peso de um pedido de reparação, m_{pedido} será, em $bytes$,

$$m_{pedido} = V_{avg}\ (bytes) \tag{3.24}$$

Em resposta ao pedido inicial de reparação o nó irá receber do nó contactado os eventos em falta caso este último os conheça. Nesta análise, assumimos que o nó contactado é capaz de responder na totalidade ao pedido inicial. Presumimos isto uma vez que consideramos que a probabilidade de ocorrerem falhas será relativamente baixa.

O tamanho da resposta depende dos eventos em falta no pedido inicial. Para avaliarmos o tamanho da resposta temos de estimar quantos eventos o nó pode ter perdido entre reparações. Considerando T_{rep} o período das reparações e T_{agg} o período das agregações de chegadas e saídas de nós, podemos estimar o número de eventos de chegada/saídas que ocorrem entre duas reparações como,

$$\frac{T_{rep}}{T_{agg}} \tag{3.25}$$

O custo associado a cada um dos eventos perdidos depende das entradas e saídas em cada evento. O valor de entradas e saídas em cada evento depende por sua vez da taxa de entrada e da taxa de saída de participantes. Uma vez que para cada nó que entrou é dado a conhecer a sua chave, endereço e chave do nó, um peso definido já anteriormente como m_k e para cada saída simplesmente é disseminada a chave do nó, m_c, assumindo uma taxa de entrada e saída iguais com o valor r, podemos estimar o peso de um evento de agregação como,

$$T_{agg} \cdot r\ (m_k + m_c) \tag{3.26}$$

Para estimar o número de eventos perdidos, temos em conta a probabilidade de perder um evento (p_{falha}), fazendo com que o número de eventos possa ser estimado da seguinte forma,

$$\frac{p_{falha} \cdot T_{rep}}{T_{agg}} \tag{3.27}$$

Uma resposta a um pedido inicial de reparação contém os eventos em falta, e a vista do segundo nó. Sabendo que o peso de um evento é dado por 3.26 e o número de eventos perdidos dado por 3.27 esta resposta terá o peso $m_{resposta}$, em bytes, com o valor de,

$$m_{resposta} = V_{Avg} + p_{falha} \cdot T_{rep} \cdot r\,(m_k + m_c)\ \ (bytes) \tag{3.28}$$

A esta resposta o nó inicial responde com os filtros em falta ao segundo nó. O peso dessa mesma resposta, $m_{resposta_2}$, tem o valor,

$$m_{resposta_2} = p_{falha} \cdot T_{rep} \cdot r\,(m_k + m_c)\ \ (bytes) \tag{3.29}$$

Deste modo, relembrando que o processo de reparação ocorre entre dois nós numa sucessão pedido-resposta-resposta, o custo total de *upload* associado a uma reparação do ponto de vista do nó iniciador, é dado, em bytes, por,

$$u_{rep_k} = m_{pedido} + m_{resposta_2} \tag{3.30}$$

$$u_{rep_k} = V_{Avg} + p_{falha} \cdot T_{rep} \cdot r\,(m_k + m_c)\ \ (bytes) \tag{3.31}$$

Com base na mesma ideia, verificamos, em termos de *download*,

$$d_{rep_k} = m_{resposta} \tag{3.32}$$

$$d_{rep_k} = V_{Avg} + p_{falha} \cdot T_{rep} \cdot r\,(m_k + m_c)\ \ (bytes) \tag{3.33}$$

Tendo em conta que, em média, um nó participa numa reparação como iniciador tantas vezes como nó contactado, verifica-se em termos de largura de banda média despendida,

$$\overline{b_u} = \overline{b_d} = \frac{u_{rep_k} + d_{rep_k}}{T_{rep_k}} \tag{3.34}$$

$$\overline{b_u} = \overline{b_d} = \frac{2 \cdot V_{Avg} + 2\left[p_{falha} \cdot T_{rep} \cdot r\,(m_k + m_c)\right]}{T_{rep_k}}\ (bytes/s) \tag{3.35}$$

Acrescentando o valor m_h do *overhead* associado ao uso do protocolo tcp/ip obtemos,

$$\overline{b_u} = \overline{b_d} = \frac{2 \cdot V_{Avg} + 2\left[p_{falha} \cdot T_{rep} \cdot r\left(m_k + m_c\right)\right] + 3m_h}{T_{rep_k}} \; (bytes/s) \tag{3.36}$$

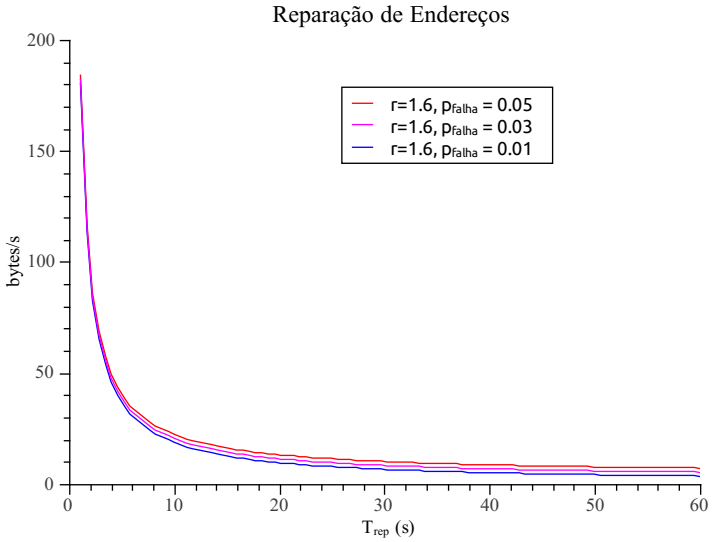

Figura 3.4: Largura de Banda despendida na reparação de endereços em função do período de reparação

Analisando a figura 3.4 podemos verificar a influência do período de agregação. Para valores demasiado pequenos, a reparação é demasiado frequente e por consequência a largura de banda despendida aumenta. Este custo na prática diz respeito aos cabeçalhos das mensagens uma vez que sendo reduzido o tempo entre reparação existe uma menor probabilidade de terem ocorrido falhas nesse intervalo de tempo. A partir de certos valores, verifica-se que a alteração da largura de banda despendida é mínima e podemos assim escolher um valor que seja suficiente para diluir o *overhead* dos cabeçalhos mas que não seja demasiado elevado, o que aumentaria o tempo de convergência.

3.5.4.2 Reparação de Filtros

A reparação de filtros ocorre em paralelo com a reparação de endereços e tem por objectivo recuperar de falhas que tenham ocorrido ao nível da disseminação de filtros.

Um pedido de reparação de filtros transportará o conjunto de filtros que o nó considera ter em falta. Deste modo, sendo p_{falha} a probabilidade de perder a disseminação de um filtro, o número de filtros em falta no momento da reparação dependerá do período de reparação T_{rep_f}, da fracção da vizinhança do nó, v e da taxa de entrada de nós no sistema r e terá o valor de,

$$n_{f_{falta}} = p_{falha} \cdot v \cdot r \cdot T_{rep_f} \tag{3.37}$$

Recordando o mecanismo de reparação de filtros descrito na secção 3.3, um nó só pode reparar parte da sua vizinhança através de outro nó uma vez que todas as vizinhanças são diferentes pois dependem da chave do nó. Se o nó escolhido aleatoriamente como alvo da reparação for um sucessor isso significa que apenas pode reparar através desse nó o conjunto de sucessores pertencentes à sua vizinhança e analogamente se o nó for um predecessor. Desta forma estimamos o peso em *bytes* de um pedido inicial de reparação de filtros seja, sendo m_{fk} o peso da chave de um filtro como,

$$m_{pedido} = \frac{n_{f_{falta}} \cdot m_{fk}}{2} \ (bytes) \tag{3.38}$$

Por seu lado a resposta a este pedido, assumindo que a reparação é sempre possível, transportará os filtros que faltam ao nó que efectuou o pedido inicial. O segundo nó enviará também o conjunto dos filtros que determina como tendo em falta e que pertencem ao intervalo especificado para que o nó inicial possa responder com esses filtros. O peso da resposta ao pedido inicial $m_{resposta}$, tendo em conta o peso de um filtro m_f, é dado pelo peso dos filtros em falta juntamente com o conjunto de filtros em falta do segundo nó,

$$m_{resposta} = \frac{n_{f_{falta}}}{2} \cdot (m_f + m_{fk}) \ (bytes) \tag{3.39}$$

O nó inicial responde finalmente a esta resposta com os filtros em falta fazendo com que o peso da resposta final, $m_{resposta_2}$ tome a forma,

$$m_{resposta_2} = \frac{n_{f_{falta}}}{2} \cdot m_f \ (bytes) \tag{3.40}$$

O custo associado a uma reparação de filtros, do ponto de vista do nó iniciador, será então,

$$u_{rep_f} = m_{pedido} + m_{resposta_2} = \frac{n_{f_{falta}} \cdot m_{fk}}{2} + \frac{n_{f_{falta}}}{2} \cdot m_f \tag{3.41}$$

$$d_{rep_f} = m_{resposta} = \frac{n_{f_{falta}}}{2} \cdot (m_f + m_{fk}) \tag{3.42}$$

Finalmente, considerando que em média um nó participa numa reparação como iniciador tantas vezes como nó contactado, podemos concluir que o custo em termos de

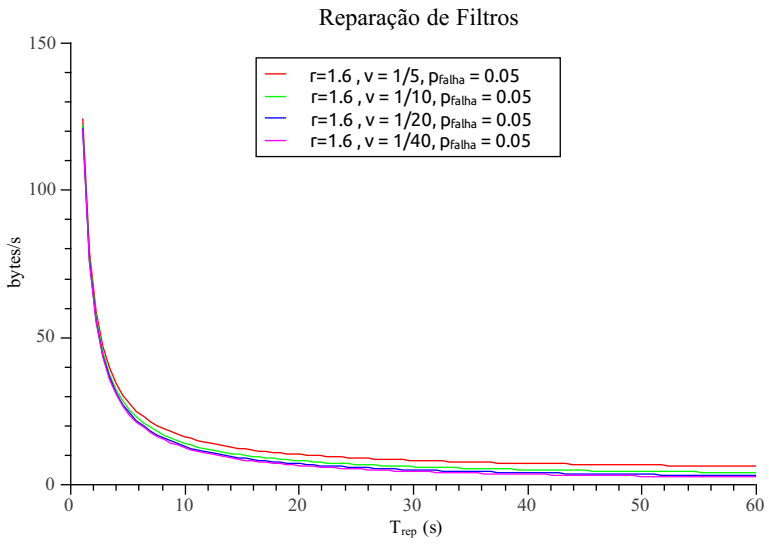

Figura 3.5: Largura de Banda despendida na reparação de filtros em função do período de reparação

largura de banda associada a uma reparação de filtros já tendo em conta o *overhead* associado ao protocolo de transporte,m_h, em bytes/s é dado por,

$$\overline{b_u} = \overline{b_d} = \frac{u_{rep_f} + d_{rep_f} + 3m_h}{T_{rep_f}} \tag{3.43}$$

$$\overline{b_u} = \overline{b_d} = \frac{\frac{n_{f_{falta}} \cdot m_{fk}}{2} + \frac{n_{f_{falta}}}{2} \cdot m_f + \frac{n_{f_{falta}}}{2} \cdot (m_f + m_{fk}) + 3m_h}{T_{rep_k}} \; (bytes/s) \tag{3.44}$$

$$\overline{b_u} = \overline{b_d} = \frac{n_{f_{falta}} \cdot (m_f + m_{fk}) + 3m_h}{T_{rep_k}} \; (bytes/s) \tag{3.45}$$

$$\overline{b_u} = \overline{b_d} = \frac{p_{falha} \cdot v \cdot r \cdot T_{rep_f} \cdot (m_f + m_{fk}) + 3m_h}{T_{rep_k}} \; (bytes/s) \tag{3.46}$$

Analisando a figura 3.5 verificamos como seria de esperar que quanto maior a vizinhança do nó maior o custo associado à reparação de filtros. No entanto, as diferenças não são muito significativas, devido à existência do *overhead* dos cabeçalhos em conjunto com uma probabilidade de falhas reduzida. À semelhança das conclusões referidas na secção anterior, a escolha do período de reparação é importante no que toca a obter o melhor compromisso entre a diluição dos cabeçalhos das mensagens e o tempo de recuperação das falhas.

3.5.5 Algoritmo de Disseminação

Na versão anterior, apoiado numa visão completa do sistema existente em cada nó, facilmente um intervalo podia ser processado por qualquer nó, uma vez que todos conhecem

os filtros necessários para realizar esse trabalho. Dessa forma, os nós facilmente garantiam que não entregavam eventos a nós que não os desejassem. Além disso, podiam também assumir que ao receberem um intervalo esse já teria sido processado por outro nó até à sua chave reduzindo assim o intervalo a tratar.

Na nova versão do sistema Livefeeds tivemos de adaptar o algoritmo de disseminação filtrada em função da criação da noção de vizinhança do nó. Uma vez que os nós participantes apenas conhecem filtros na sua vizinhança, ao receber um intervalo para processar o nó poderá ter de entregar o evento a um nó fora da sua vizinhança não sabendo se este aceita ou não o evento. Ao receber um intervalo o nó não pode assumir que os nós anteriores à sua chave foram tratados uma vez que o nó do qual recebeu o intervalo podia não conhecer os filtros desses nós. Por esta razão, as mensagens tem de registar o seu percurso de modo a evitar duplicados. Este registo irá aumentar o peso da mensagem à medida que a disseminação avança o que por sua vez aumenta o custo algoritmo de disseminação em relação à versão anterior. O custo deste registo m_r é dado em função do nível da árvore de disseminação, sendo m_c é o custo associado a guardar registo de dado nó como,

$$m_r(i) = m_c \cdot i \tag{3.47}$$

Adicionando o custo do envio do evento ou notificação, m_e, e sabendo que a cada nível são efectuados G envios pelo nó, obtemos o custo de *upload* total u em função do nível da árvore,

$$u_i = G \times (m_e + m_c \cdot i) \tag{3.48}$$

O custo médio do *upload* realizado por um nó durante uma disseminação será,

$$\overline{u} = \frac{\sum_{i=0}^{(h-1)} G\,(m_e + m_k \cdot i)}{(h-1)} \tag{3.49}$$

$$\overline{u} = \frac{G\left[m_e\,(h-1) + \frac{m_k(h-1)\cdot h}{2}\right]}{(h-1)} \tag{3.50}$$

$$\overline{u} = G\left(m_e + \frac{m_k \cdot h}{2}\right) \tag{3.51}$$

Em termos de *upload* apenas os nós interiores (nós que não são folhas) irão realizar trabalho e por isso, tendo em conta, como demonstrado em estudo prévio do sistema Livefeeds [22], a probabilidade de um nó ser interior é de $\frac{1}{G}$, teremos um custo médio de largura de banda despendida em *upload* por disseminação de,

$$\overline{u} = m_e + \frac{m_k \cdot h}{2} \tag{3.52}$$

Por outro lado a largura de banda despendida em *download*, uma vez que cada nó recebe apenas uma mensagem durante a disseminação, terá o mesmo valor,

$$\overline{d} = \overline{u} = m_e + \frac{m_k \cdot h}{2} \tag{3.53}$$

Acrescentando o valor m_h dos *headers* tcp/ip obtemos,

$$\overline{d} = \overline{u} = m_e + \frac{m_k \cdot h}{2} + m_h \tag{3.54}$$

A altura das disseminações é função do número de participantes do sistema, valor que depende taxa de entrada de participantes e do tempo médio de sessão.

$$N_{avg} = T_{session} \times r \tag{3.55}$$

Se todos os nós participarem numa disseminação podemos estimar, tendo em conta que as disseminações são árvores de grau G, o valor de h,

$$h \approx log_G(N_{avg}) = log_G(T_{session} \times r) \tag{3.56}$$

Como já foi referido na secção 2.4.7 este custo é influenciado pela largura dos filtros dos nós. Isto é, se os participantes possuírem filtros que aceitam poucos eventos, os nós que participam nas disseminações serão em menor número, resultando em árvores de menor altura e um custo médio mais pequeno. Por outro lado se os nós possuírem filtros que aceitam muitos eventos, haverá mais nós a participarem na disseminação, resultando numa altura maior e num custo médio mais elevado. Os filtros dos participantes influenciam deste modo a popularidade do evento. Se a popularidade média dos eventos do sistema for $\frac{1}{2}$, isto significa que em média metade dos participantes devem receber um evento quando este ocorre pois o seu filtro aceita esse evento. Isto implica que a taxa média de eventos em cada nó é metade da taxa de eventos gerados no sistema.

Recordamos também que durante o algoritmo de disseminação filtrada, com o intuito de evitar falsos negativos, os nós enviam as suas vistas. Acrescentando o peso médio da vista de eventos podemos estimar o custo médio de largura de banda despendida por cada nó em bytes/segundo como,

$$\overline{b_u} = \overline{b_d} = \left[V_{avg} + m_e + \frac{m_c \cdot log_G \left(T_{session} \times r \times pop \right)}{2} + m_h \right] \times r_e \times pop \ (bytes/s) \quad (3.57)$$

Este seria o valor esperado se os nós que participassem nas disseminações fossem estritamente aqueles que aceitam o evento. No entanto, a nova versão do livefeeds tem associada um problema que chamamos de falsos positivos e implica justamente que existem nós que podem receber eventos cujo filtro não o aceita.

3.5.6 Introdução de Falsos Positivos

Nas secções acima demonstrámos o benefício de uma redução da visibilidade que cada participante possui em relação ao resto do sistema. Esta redução, em termos do conhecimento dos filtros, resultou numa diminuição do custo de largura de banda associado à entrada de cada nó. Desta forma, taxas de chegada que na versão anterior eram consideradas demasiado dispendiosas são agora mais facilmente acomodadas pelo sistema Livefeeds. O sistema torna-se deste modo mais resistente a picos de afluência.

Apesar do benefício claro em termos de largura de banda resultante da nova versão do algoritmo de filiação, a redução da visibilidade dos nós dá origem a um problema que não ocorria na versão anterior. Os nós podem agora, durante a disseminação de um evento, ser solicitados a processar nós que não pertencem à sua vizinhança. Nesta situação, não conhecendo o filtro dos nós, existe a possibilidade de gerar um falso positivo ao entregar o evento a dado nó.

3.5.6.1 Vizinhança do Nó e Grau das Disseminações

Nas simulações efectuadas procurámos verificar a existência de falsos positivos e quantificar a sua ocorrência em função dos diversos parâmetros. O primeiro parâmetro que podemos prever que tenha uma grande influência na introdução de falsos positivos é o tamanho da vizinhança do nó. Quanto menor a vizinhança do nó, menor o número de nós conhecidos pelo nó e por isso maior a probabilidade de processar nós fora do seu intervalo de vizinhança.

Durante o processo de disseminação de um evento por um dado intervalo (inicialmente igual ao espaço de chaves) existe uma subdivisão deste intervalo em G subintervalos determinando assim o grau e por consequência a altura da árvore de disseminação. À medida que os subintervalos se tornam cada vez mais pequenos diminui a probabilidade de serem gerados falsos positivos até que estes passam a estar contidos na vizinhança, altura em que a probabilidade de gerar um falso positivo será nula.

O número de falsos positivos gerados a cada nível da disseminação depende do grau G da árvore, que determina o número de envios feitos a cada nível, e da vizinhança do nó. Inicialmente a vizinhança do nó será a fracção v do espaço global e a raiz começa por efectuar G envios. O número médio de falsos positivos gerados no primeiro nível será

então de,

$$\overline{f_0} = G(1 - v) \tag{3.58}$$

Nos níveis seguintes o número de nós que realizam trabalho aumenta em função da subdivisão dos intervalos por G, sendo que o nível i são realizados G^i envios. Considerando K_0 como o número de chaves possíveis no espaço de chaves do sistema verificamos que à medida que aumenta o nível da disseminação o número de chaves possível no intervalo a tratar, K_i, evolui da acordo com a expressão abaixo.

$$K_i = \frac{K}{G^i} \tag{3.59}$$

Desta forma, a fracção da vizinhança (o número de chaves possíveis na vizinhança do nó face ao número de chaves possíveis no espaço global de chaves) do nó em relação ao intervalo a tratar também diminui, tomando a forma,

$$v_i = \frac{K \cdot v}{K_i} = \frac{K \cdot v \cdot G^i}{K} = v \cdot G^i \tag{3.60}$$

Assim o número médio de falsos positivos gerados por cada nó, é à semelhança do que passa na raiz,

$$\overline{f_i} = G(1 - v_i) = G(1 - v \cdot G^i) \;\; , \;\; i < log_G(\frac{1}{v}) \tag{3.61}$$

À medida que i aumenta, diminui a probabilidade de ocorrem falsos positivos até que é nula quando os intervalos ficam contidos na vizinhança do nó ($v \cdot G^i = 1$). O número total de falsos positivos em cada nível, atendendo ao facto de que existem G^i nós no nível i terá o valor de acordo com a expressão abaixo.

$$\overline{f_i} = \begin{cases} G^i \cdot G(1 - v \cdot G^i) & , \;\; i < log_G(\frac{1}{v}) \\ 0 & , \;\; i \geq log_G(\frac{1}{v}) \end{cases} \tag{3.62}$$

O comportamento descrito, é apresentado, para diferentes vizinhanças, na figura 3.6 abaixo usando como exemplo $G = 2$.

Desde já verificamos que só os níveis mais perto da raíz tendem a gerar falsos positivos mesmo para um valor de G reduzido. Quanto maior for G, mais rapidamente os subintervalos ficarão contidos na vizinhança dos nós, reduzindo os níveis que geram falsos positivos. Ainda assim, um maior número de nós irá realizar envios nesses primeiros

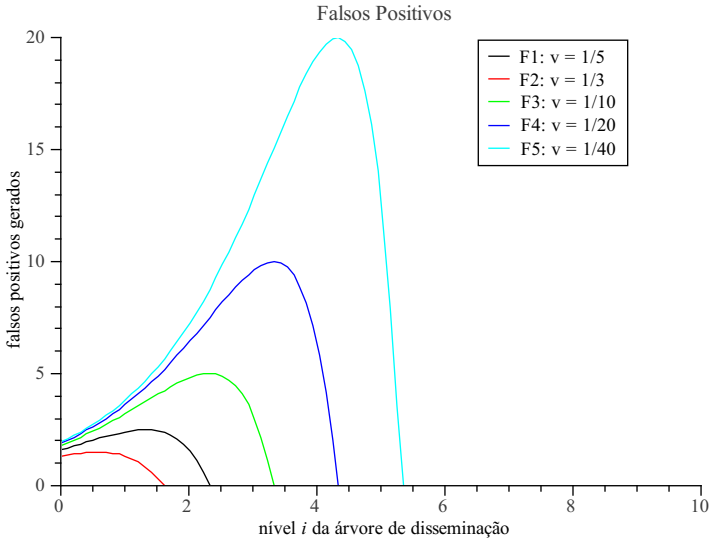

Figura 3.6: Falsos positivos gerados por nível de árvore de disseminação ($G = 2$)

níveis. Interessa-nos então perceber de que forma o total de falsos positivos se comporta para diferentes valores de G. Usando a expressão 3.62, podemos calcular o total de falsos positivos que ocorrem numa disseminação, N_{fp}, somando o número de falsos positivos que são gerados em cada nível.

$$N_{fp} = \sum_{i=0}^{log_G\left(\frac{1}{v}\right)-1} G^i \cdot G(1 - v \cdot G^i) = \frac{G \cdot (v-1)\,(v-G)}{(G^2-1) \cdot v} \tag{3.63}$$

No gráfico 3.7 apresentamos as curvas do número total de falsos positivos em função de G.

Podemos observar o comportamento descrito acima. À medida que G aumenta o número total de falsos positivos diminui. No entanto, esta diminuição é contrariada pelo aumento de envios que ocorrem nos primeiros níveis enquanto os intervalos não estão contidos nas vizinhanças. Esta diminuição é ainda tanto menor quanto maior for a vizinhança do nó, uma vez que mesmo para um grau pequeno rapidamente os intervalos ficam contidos nas vizinhanças. Isto indica-nos que para cada vizinhança existe um valor diferente de G em que obtemos o melhor compromisso em termos de falsos positivos gerados e balanceamento de carga entre os participantes.

Notamos também o reduzido número de falsos positivos gerados que, comparativamente com o número de participantes nos sistemas, fazem prever valores de falsos positivos percentualmente baixos.

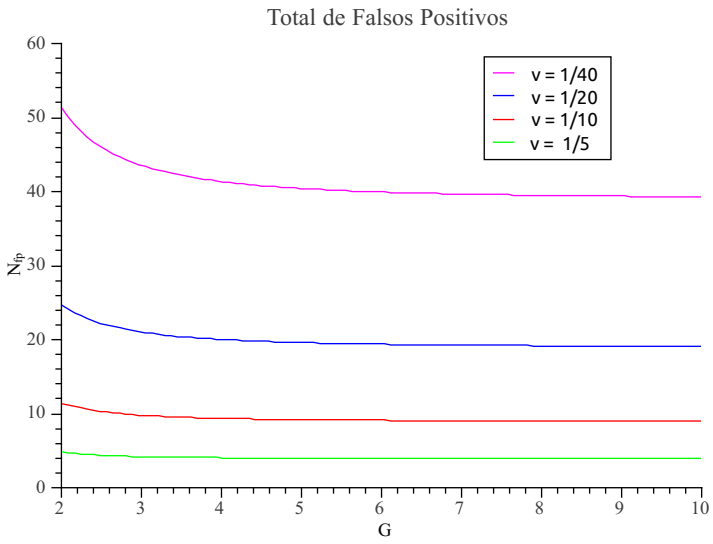

Figura 3.7: Falsos positivos gerados por nível de árvore de disseminação

3.5.6.2 Falsos Positivos e Taxa de Entrada de Participantes

Na figura secção anterior verificámos que a partir de certo nível de uma árvore de disseminação não serão gerados quaisquer falsos positivos. Isto significa que a partir deste valor a altura da árvore não influencia o número de falsos positivos, ou seja para uma mesma vizinhança v e grau G, duas disseminações de altura h_1 e h_2 geram o mesmo número médio de falsos positivos caso se verifique,

$$h_1 \geq h_2 \geq log_G\left(\frac{1}{v}\right) \tag{3.64}$$

Se todos os nós participassem na disseminação teríamos árvores com a altura de aproximadamente,

$$h \approx log_G(N_{avg}) = log_G(T_{session} \times r) \tag{3.65}$$

Usando parte da expressão 3.64 obtemos a condição que determina a partir de que taxa de chegada o número médio total de falsos positivos não se altera,

$$log_G(T_{session} \times r) \geq log_G\left(\frac{1}{v}\right) \tag{3.66}$$

$$r \geq \frac{1}{v \times T_{session}} \tag{3.67}$$

Isto leva-nos a concluir que são necessárias taxas de entrada muito reduzidas para que

esta tenha influência directa na totalidade de falsos positivos gerados por disseminação. Taxas de tal forma reduzidas vão contra aquela que é a premissa e motivação da nossa solução - suportar elevadas taxas de entrada. Ainda assim nem todas as disseminações terão a mesma altura uma vez que temos de ter em conta a popularidade dos eventos que ocorrem no sistema. Este factor é discutido em maior de detalhe na secção seguinte.

3.5.6.3 Popularidade dos Eventos

Na secção anterior concluímos que seriam necessárias taxas de entrada bastante reduzidas para que este fosse um factor que influenciasse o número total de falsos positivos. Esta conclusão baseou-se no cálculo da altura média das árvores de disseminação. No entanto, assumimos que todos os nós participavam na disseminação, mas uma vez estamos a discutir uma disseminação filtrada o que esperamos é que para cada evento apenas uma parte dos nós participem na disseminação. Desta forma, para cada evento apenas uma fracção de nós participa na árvore. Esta fracção é o número de nós que aceitam o evento face à totalidade dos participantes e pode ser também designada como a popularidade de um evento. Assim, a altura média das árvores de disseminação será mais baixa que o referido.

Quanto menor for a popularidade, menor é o número de nós que realmente aceitam o evento. Devido à noção de visibilidade parcial, em valores de popularidades muito reduzidos existe um grande *overhead* em termos de falsos positivos comparativamente ao número de participantes que no final da disseminação devem realmente receber o evento quando os intervalos ficam contidos nas vizinhanças. Isto implica que a popularidade média dos eventos é um facto que influencia a percentagem de falsos positivos que ocorrem num sistema. Caso a popularidade média seja baixa, existe um grande número de disseminações nas quais o *overhead* é elevado. Por outro lado, se a popularidade média for elevada isto significa que o número de participantes que aceitam um evento é mais próximo do número de nós que devem receber o evento e como tal o *overhead* de falsos positivos é relativamente menor.

Na figura 3.8 podemos verificar o comportamento descrito acima. Quanto menor a vizinhança, maior o impacto dos falsos positivos. Ainda assim, este impacto ocorre maioritariamente para valores de baixa popularidade. Quanto mais popular um evento, menor o impacto da vizinhança. Na prática os valores de falsos positivos para popularidades altas são ainda menores pois existe uma maior probabilidade de que o filtro do nó contactado às cegas aceite à mesma o evento. Podemos concluir assim, que a escolha do tamanho de vizinhança deve ter em conta, se possível, a popularidade média esperada num sistema.

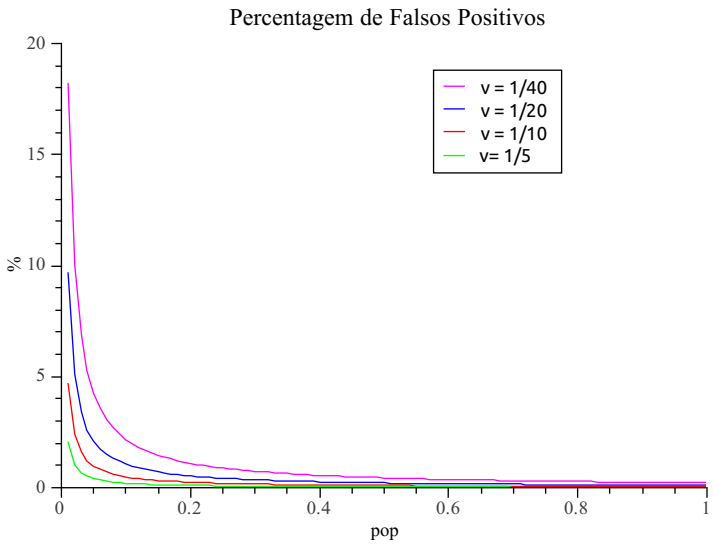

Figura 3.8: *overhead* de falsos positivos numa disseminação face à popularidade do evento

3.5.6.4 Falsos positivos, a Altura das Árvores e o custo das Disseminações filtradas

Na secção 3.5.5 demonstrámos o custo médio esperado de *download* e *upload* em termos de largura de banda do algoritmo de disseminação filtrada. Concluímos que o custo do algoritmo de disseminação filtrada depende da popularidade média dos eventos uma vez que este determinará a altura das árvores de disseminação, uma vez que esta guarda registo dos nós que a encaminharam em cada nível. No entanto, a nossa estimativa da altura das árvores de disseminações assume que apenas os nós cujo filtro aceita o evento participavam na disseminação. Como vimos a discutir até aqui, este não é o caso com a nova versão do sistema Livefeeds, uma vez que esta sofre com a possível introdução de falsos positivos.

A introdução de falsos positivos nas árvores de disseminação filtrada implica que a altura destas será maior do que o valor esperado caso estes não ocorressem. Recordemos que, na secção 3.5.6.1 concluímos que o maior número de falsos positivos ocorre no início da disseminação quando a probabilidade da sua ocorrência é maior pois o intervalo a tratar é ainda consideravelmente maior que a vizinhança dos nós. Este facto revela que a introdução de falsos positivos é um problema não pelo o número de ocorrências (que se provou ser baixo em relação ao número total de envios) mas sim pelas circunstâncias em que estes ocorrem. Ao ocorrerem maioritariamente no início das disseminações os falsos positivos aumentam a altura das árvores e relegam os nós que realmente aceitam os eventos para níveis inferiores onde estes irão pagar uma factura maior pela recepção do evento em termos de custo de *download/upload* e latência.

A ocorrência dos falsos positivos mais cedo nas disseminações é um problema para os nós que numa particular disseminação aceitam o evento. No entanto, é um facto que suaviza o impacto num nó que está a realizar trabalho para disseminar um evento que

não deveria passar por ele em primeiro lugar. Na prática, isto resulta que os nós estão a pagar nas disseminações de eventos que estão interessados aquilo que não pagam quando são chamados a trabalhar numa disseminação de um evento como resultado de um falso positivo. Este efeito será tão mais notório quanto menor for a popularidade média dos eventos, pois para uma popularidade média alta a altura média das árvores já seria elevada e existe menor probabilidade de ocorrerem falsos positivos.

Uma forma de suavizar este problema seria aumentar o grau da árvore. Um grau reduzido implica uma altura maior e na ocorrência falsos positivos o seu impacto é intuitivamente muito maior nos custos associado ao registo do percurso da mensagem. Se aumentarmos o grau da árvore podemos diminuir este impacto e por sua vez o custo que os nós que realmente aceitam o evento vão ter na disseminação. Devemos, no entanto, como já foi referido, ter em conta que um aumento do grau da árvore resulta possivelmente num pior balanceamento da carga das disseminações.

3.5.6.5 Redundância de Filtros

Na secção 3.4.2 introduzimos o conceito de redundância de filtros no sistema Livefeeds. Podemos definir a redundância de filtros como a percentagem de filtros que não são únicos no sistema. Sabendo que existem N participantes no sistema, e considerando F_u o número de filtros únicos no sistema. O valor da redundância, ρ,para esse sistema é dado pela expressão abaixo.

$$\rho = 1 - \frac{F_u}{N} \tag{3.68}$$

A existência de filtros repetidos pode ser usada no algoritmo de disseminação filtrada. Quando chamados a tratar um intervalo que não pertence à sua vizinhança, sacrificando tempo de computação, os nós podem percorrer toda a parte do intervalo a tratar verificando, através da chave dos filtros desses nós, se os filtros são repetidos face a um filtro pertencente à sua vizinhança. Caso seja encontrado um nó cujo filtro seja redundante em relação a um filtro do seu conhecimento, o nó pode testar o filtro e caso este aceite o evento entregar o evento a esse nó evitando um falso positivo.

A probabilidade de evitar um falso positivo depende do valor da redundância que se verifica no sistema. Quanto maior for a redundância, mais filtros o nó conseguirá testar e como tal maior a probabilidade de encontrar um nó que aceite o evento. A vizinhança do nó influencia esta probabilidade na medida em que, quanto mais filtros o nó conhece maior a probabilidade de conhecer o filtro redundante de um nó exterior à sua vizinhança. Ao mesmo tempo existe uma grande influência da popularidade. Se o evento for pouco popular haverá menor probabilidade de encontrar um nó que aceite o evento e por isso mais difícil será evitar o falso positivo.

4

Protótipo e Validação Experimental

4.1 O Ambiente de Simulação

Os testes e a validação dos algoritmos descritos foram realizados num ambiente de simulação. Este simulador, implementado usando a linguagem Java, simula uma infraestrutura comparável à rede real e permite simular com facilidade a presença de múltiplos nós e as suas respectivas interacções. Para além de simular a execução do sistema, o simulador permite monitorizar a simulação durante a sua execução bem como, a geração de estatísticas que podemos usar para facilmente criar gráficos em tempo real ou numa altura posterior. É possível também, a criação de várias configurações de testes que podem ser usadas em regime *batch* para simulações de carácter mais intensivo.

A grande vantagem do uso de um simulador é a redução da fase de testes uma vez que o *deployment* é muito mais simples do que se o sistema fosse testado na rede real. Deste modo, conseguimos um ágil ciclo de desenvolvimento em que os algoritmos podem ser facilmente afinados de acordo com resultados preliminares.

4.1.1 Modelos de Churn

O sistema Livefeeds à semelhança da maior parte dos sistemas que usam redes *peer-to-peer* é um sistema dinâmico. Isto significa que o número de participantes no sistema não é fixo, mas sim variável em função da entrada e saída de nós. Este dinamismo, em termos da taxa de chegada de participantes e a duração da sua estadia no sistema, é muitas vezes designado por *churn*.

Quando procedemos à validação experimental de um sistema em que se espera alto

dinamismo numa situação real, é vital conseguir reproduzir de forma o mais aproximadamente possível, o comportamento dos seus participantes. Para este efeito usamos um modelo de *churn*.

Um modelo de *churn* é caracterizado por uma distribuição que modela o intervalo entre chegadas consecutivas de novos nós e uma distribuição que modela o tempo de permanência de um nó no sistema (tempo de sessão).

O sistema Livefeeds não apresenta uma taxa de chegadas de participantes constante. Sendo uma plataforma de suporte à distribuição de conteúdos, a taxa de chegada de novos participantes pode ser afectada pela ocorrência de eventos que podem gerar uma grande afluência de participantes ao sistema. O uso do sistema varia também consoante as horas do dia e dias da semana, sendo previsível que haja uma maior afluência nos períodos diurnos. De forma geral é possível criar um paralelismo entre o comportamento dos utilizadores do Livefeeds e dos participantes do sistema *Skype*. Estudo detalhado do sistema Skype [15] e outros relativos às redes *peer-to-peer* em geral [30], sugerem que tal comportamento pode ser aproximado modelando a chegada de participantes com base num processo de Poisson não-homogéneo e as suas sessões com base numa distribuição *heavy-tailed*. Uma distribuição que apresenta boas condições de se aplicar ao sistema Livefeeds para a modelação dos tempos de sessão é a distribuição de *Weibull*.

4.1.2 Representação dos Participantes

No simulador os participantes de um sistema são representados na sua forma mais básica pela classe *AbstractNode*. Esta classe define o esqueleto básico de um nó participante de um sistema distribuído. O principal aspecto é aquele define um *AbstractNode* como um *handler* de mensagens através da interface *MessageHandler*. Ao estender a classe *AbstractNode* podemos implementar diferentes tipos de nós para simular diferentes tipos de sistemas. Na nossa implementação fazêmo-lo através da classe *CatadupaNode* que implementa os atributos e a representação do estado de um nó do sistema Livefeeds. As diferentes mensagens que os nós podem receber são então definidas por uma interface que estende a classe *MessageHandler*. Esta interface é depois implementada pela classe *CatadupaNode* onde são defindos os procedimentos a tomar quando o nó recebe cada tipo de mensagem.

4.1.3 Filtros e Eventos

O aspecto central da disseminação filtrada são os filtros dos participantes e os eventos que irão ser avaliados por estes filtros. Implementamos estas noções de forma extremamente simples e eficiente ao dizermos que um filtro é representado por um valor limite no intervalo $[0, 1]$. Um evento é caracterizado igualmente por um valor nesse intervalo. Um filtro de valor 0 não aceita qualquer evento enquanto que um filtro de valor 1 aceita todos os eventos. Um filtro de valor f irá aceitar todos os eventos tal que,

$$\{ev.value\} \leq f \tag{4.1}$$

4.1.3.1 Redundância de Filtros

Normalmente o filtro de cada nó seria atribuído aleatoriamente mas um dos aspectos que queremos estudar está directamente relacionado com a redundância de filtros. Deste modo, desenhámos uma classe *FilterDispenser* que atribui filtros aos nós que nascem consoante o valor alvo de redundância definido para a simulação em questão.

4.1.4 Representação da Base de Dados dos Participantes

Para cada nó simulado é necessário representar a sua base de dados. Isto coloca-nos um problema em termos de como representar a base de dados num sistema que irá atingir uma dimensão considerável. Obviamente não é prático representar individualmente a base de dados de cada participante. Em alternativa, implementamos um mecanismo de base de dados global baseado em máscaras. Para cada nó que nasce no sistema é lhe atribuído pelo simulador um número de sequência com base num contador. Em cada nó é então representado o seu conhecimento dos participantes através de uma estrutura baseada num *bitset*. Esta estrutura é definida por uma base b e uma palavra binária representando os nós conhecidos. Definido o tamanho em bits do da palavra, cada vez que todos os bits são positivos a base é incrementada por esse valor. Por exemplo, para um valor $b = 4$ o valor 1101 significaria que o nó em questão conhece os nós até 4 inclusive bem como os nós 5, 7 e 8 mas desconhece o nó 6. Os nós aplicam então esta máscara à base de dados global como forma de acederem à sua própria base de dados. O uso desta estratégia, baseada no deslocamento dos *bitsets* permite reduzir a memória necessária a representação das base de dados.

4.1.5 Vistas de Eventos de Agregação de Entradas/saídas

Um aspecto também importante da implementação diz respeito à representação das vistas de eventos de agregação dos sequenciadores. Estas vistas são implementadas recorrendo à mesma estrutura descrita na secção anterior baseando-se no facto de que cada evento de agregação é identificado por um número de sequência. Os eventos de agregação são então guardados, para efeitos de simulação, numa base de dados global e não individualmente em cada nó pois tal seria extremamente redundante e um grande desperdício de memória. Quando um nó necessita aceder a um destes eventos fá-lo através da base de dados global caso se verifique na sua vista que o conhece.

4.1.6 Vistas de Filtros

Com a nova versão do Livefeeds em que separámos a disseminação de filtros dos restantes atributos dos nós é necessário representar separadamente a vista de filtros uma vez

que o nó pode conhecer o endereço de um nó e não conhecer o seu filtro.

Introduzindo a noção de vizinhança em função da chave do nó criamos uma situação em que todas as vistas e o conjunto de eventos de disseminação de filtros recebido são diferentes. Isto torna difícil representar o conhecimento de filtros através de uma vista de eventos ao estilo do que foi feito para os eventos de agregação. Excluindo esta hipótese, concluímos que temos de representar o conhecimento de filtros em cada nó. Poderíamos então guardar em cada nó uma lista dos filtros que este conhece mas estaríamos possivelmente a criar um problema em termos de necessidades de memória. Para sistemas que atingem um grande número de participantes guardar em cada participante uma lista de filtros conhecidos seria altamente dispendioso. Para contornar este problema realizamos, ao nível da implementação, a ideia contrária - guardamos uma lista de filtros desconhecidos. Isto é, em cada nó mantemos o conjunto de filtros que este tem em falta. Inicialmente quando o nó faz o *download* da base de dados inicial de endereços o nó preenche a sua lista de filtros em falta com os filtros que consoante essa base de dados ele irá receber. Durante o *download* da base de dados de filtros o nó vai retirando os filtros que fica a conhecer. Para nós que entram no sistema depois de um nó já ter feito o *download* da base de dados os filtros serão conhecidos através do algoritmo de disseminação de filtros. No entanto, como o novo nó não pertencia à base de dados inicial este não é dado como em falta na lista de filtros. Para resolver este problema, durante a disseminação de filtros antes de um envio de uma mensagem que contém o filtro do novo nó, o filtro é introduzido artificialmente pelo emissor na lista do receptor para que este saiba que irá receber o filtro e, em caso de falha do envio, esta seja contabilizada para a reparação.

4.1.7 Implementação dos Algoritmos

Como foi referido na secção 4.1.2 acima, o comportamento dos nós é implementado em classes que extendem a classe abstracta *AbstractNode*. Numa classe *CatadupaNode* implementamos o comportamento do nó em relação ao algoritmo de filiação. Definimos na interface *CatadupaSocketHandler* as mensagens que o nó recebe durante o algoritmo de filiação. Estas fazem um mapeamento quase directo para aquelas que são as mensagens descritas nas interacções do capítulo da concepção da solução. O algoritmo de disseminação é implementado numa classe *TurmoilNode* que extende a classe *CatadupaNode*. Esta modularidade permite-nos testar facilmente diferentes versões dos algoritmos o que seria muito mais complexo num teste em rede real.

4.1.8 Tráfego e Estatísticas

No final da implementação dos algoritmos o aspecto mais importante das nossas simulações é a contabilização do tráfego entre os participantes. Esta contabilização é a base da nossa validação experimental e das conclusões que pretendemos retirar da nova versão do sistema.

Cada mensagem no simulador é representada por uma classe que é instanciada para

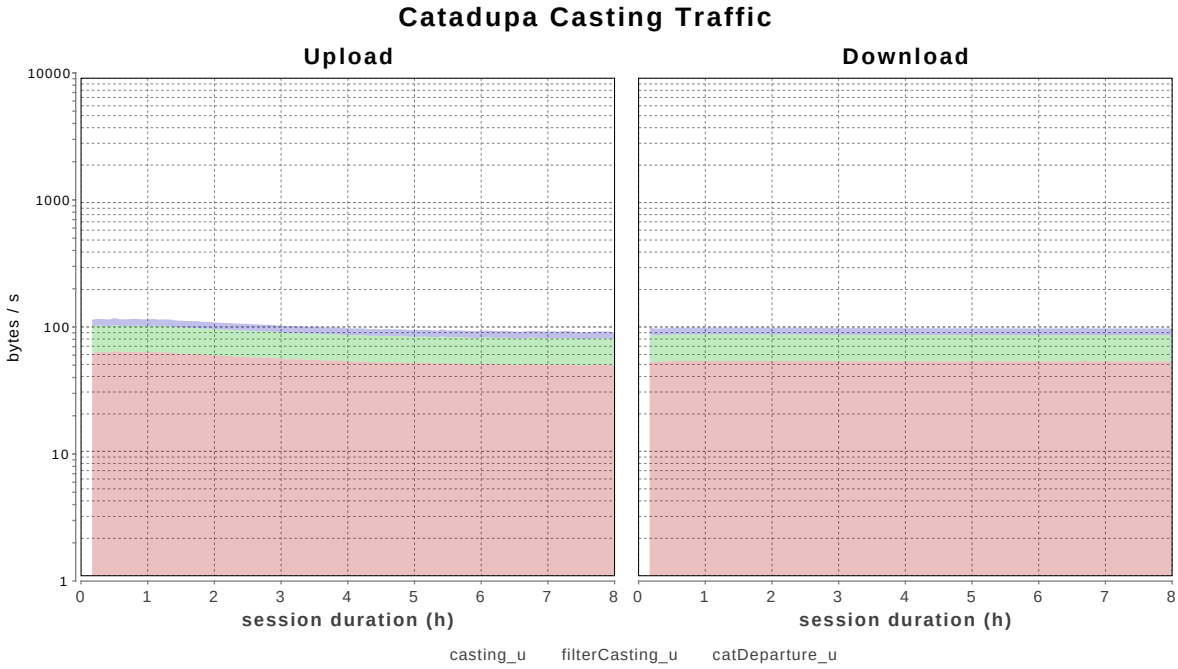

Figura 4.1: Exemplo de gráfico obtido no simulador

cada mensagem trocada entre participantes. Este objecto carrega a informação que seria transmitida pela rede real. Para cada mensagem podemos então definir o seu peso em bytes e assim durante o seu envio e recepção contabilizar entre o emissor e receptor, o tráfego em *download* e *upload*.

As contabilizações de tráfego são então usadas para gerar estatísticas. Estas estatísticas são no nosso caso contabilizações de tráfego dos diversos aspectos do sistema em função do tempo de sessão dos nós. Também criamos estatísticas de outros aspectos, como por exemplo a popularidade dos eventos, os falsos positivos gerados, a altura das árvores de disseminação entre outras.

Na figura 4.1 podemos ver um exemplo de um gráfico gerado durante uma simulação.

4.1.9 Limitações do Simulador

A principal limitação do simulador diz respeito à escala dos sistemas que este suporta. Por desenho, o simulador implementa o comportamento dos nós como tarefas que são executadas de forma sequencial. Na presença de muitos nós simulados este facto coloca um grande peso no escalonador dessas tarefas. Isto significa em termos práticos que é difícil obter resultados em tempo útil para simulações que envolvam participantes na ordem de 10^5.

4.1.10 Parâmetros de Simulação

Os testes apresentados são resultado de diferentes parametrizações da infraestrutura simulada e dos próprios algoritmos. É importante definir à partida quais aqueles que iremos estudar uma vez que as combinações possíveis seriam virtualmente infinitas, podendo facilmente perder-se o foco daqueles que são os parâmetros directamente sob estudo.

O principal objectivo desta dissertação é o estudo duma nova versão do Livefeeds em que criámos a noção de vizinhança de um nó. Esta vizinhança consiste no espaço de chaves definido em função da chave do nó para qual o nó conhece os filtros. Com a criação desta vista parcial do sistema, por oposição à anterior vista global, esperamos reduzir a largura de banda usada. No entanto, existe a possibilidade de existirem falsos positivos, uma ocorrência que queremos reduzir. Os testes efectuados visam quantificar o benefício em termos da redução de largura de banda e o compromisso em termos da introdução de falsos positivos.

A vizinhança do nó i terá a forma $[k_i - d, k_i + d]$. O parâmetro d é assim um dos principais factores em estudo. No entanto, uma das principais motivações é poder acomodar elevadas taxas de chegadas de participantes pelo que esta taxa (r) será igualmente um factor em destaque.

Finalmente, existem outros factores secundários, mas relevantes, que irão influenciar a largura de banda despendida e o número de falsos positivos introduzidos. São exemplo destes, o tamanho dos filtros, a redundância dos filtros presentes no sistema e a popularidade dos eventos disseminados (número de participantes face ao total cujo filtro aceita o evento).

Identificados os principais factores, o seu papel e os resultados em função dos mesmos serão apresentados com maior detalhe nas secções seguintes.

4.2 Resultados Experimentais

Nesta secção apresentamos os resultados obtidos através das simulações efectuadas. Usámos principalmente duas configurações base em que variamos a taxa de entrada de participantes e cada uma destas subdividindo-se em quatro configurações onde variamos a vizinhança do nó. A taxa de chegada de participantes é um factor de grande relevância para a validação da solução mas encontramo-nos limitados pelo simulador na medida em que valores muito elevados tornam a simulação demasiado pesada e é impossível obter resultados em tempo útil.

Configuração 1

1: Taxa de Chegadas, $r = 1.6$ nós/s
2: Weibull $k = 1.8$
3: Tempo Médio de Sessão $4h$
4: Tempo Máximo de Sessão $8h$
5: Peso do Filtro, $m_f = 250$ bytes
6: Peso da Chave, Endereço e Chave de Filtro, $m_e = 22$ bytes
7: Vizinhança do nó em relação ao espaço de chaves global, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{40}$
8: Grau disseminação filtrada, $G = 2$

Configuração 2

1: Taxa de Chegadas, $r = 3.2$ nós/s
2: Weibull $k = 1.8$
3: Tempo Médio de Sessão $4h$
4: Tempo Máximo de Sessão $8h$
5: Peso do Filtro, $m_f = 250$ bytes
6: Peso da Chave, Endereço e Chave de Filtro, $m_e = 22$ bytes
7: Vizinhança do nó em relação ao espaço de chaves global, $\frac{1}{20}$, $\frac{1}{40}$
8: Grau disseminação filtrada, $G = 2$

4.2.1 Descarregamento da Base de Dados Inicial

Começamos por apresentar os resultados relativos ao descarregamento da base de dados inicial. Este descarregamento divide-se no descarregamento dos endereços, chaves e chaves de filtros e o descarregamento dos filtros.

4.2.1.1 Base de Dados de Endereços, Chaves e Chaves de Filtros

No gráfico 4.2 apresentamos graficamente o resultado de uma simulação no que diz respeito aos custos relativos ao *download* e *upload* das bases de dados iniciais. Este resultado representa um padrão que se verifica graficamente para as diferentes vizinhanças sendo que a diferença verificada é apenas em termos da magnitude dos valores obtidos. Desta forma apresentamos restantes resultados de seguida em forma de tabela.

Podemos verificar no gráfico que a largura de banda despendida em *upload* vai diminuindo à medida que o nó permanece mais tempo no sistema. Isto acontece devido a existência do mecanismo de balanceamento entre o *upload* e o *download* de um nó. Quando os nós chegam ao sistema começam por fazer o descarregamento das bases de dados iniciais tendo um grande custo nos primeiros momentos de vida como podemos verificar. Isto implica também que houve nós que tiveram de realizar trabalho em termos de *upload*. Os nós existentes, cujo *upload* se encontrar balanceado com o seu *download* podem recusar um pedido de envio de base de dados levando a que outros nós, que ainda não realizaram trabalho de *upload*, tenham uma maior probabilidade de ser escolhidos. Este mecanismo tem o propósito de balancear a carga entre os participantes. Adicionalmente comprovamos graficamente a conclusão de que o *download* e *upload* tomam valores

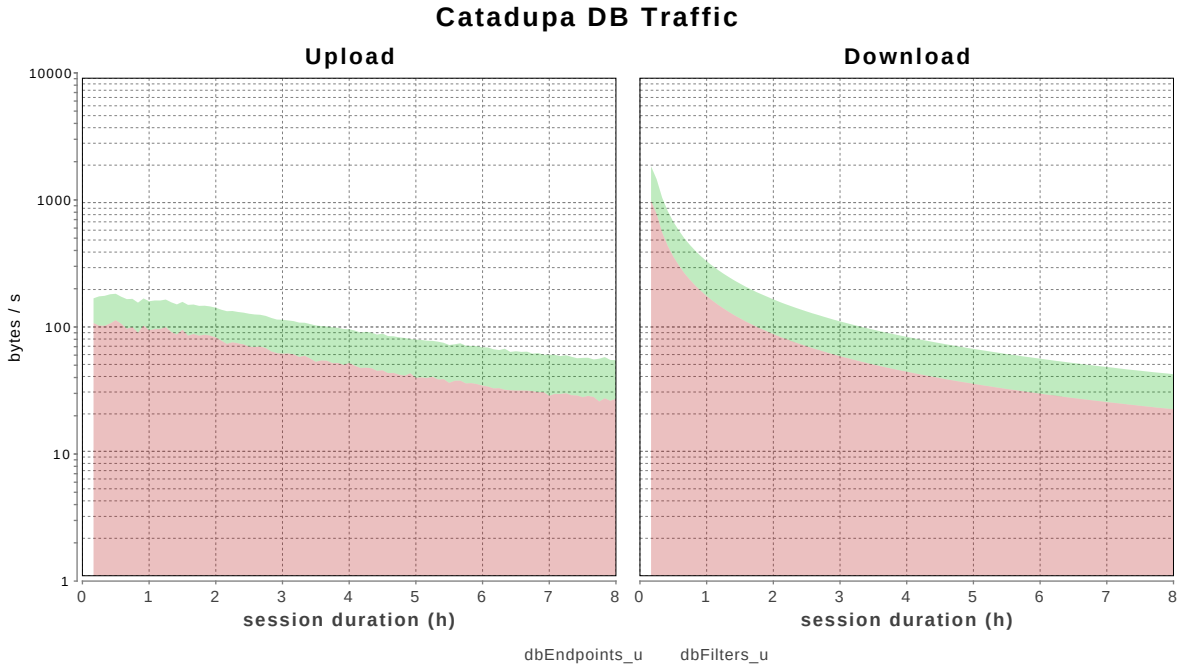

Figura 4.2: Custo do descarregamento da base de dados inicial $r = 1.6$, $v = \frac{1}{10}$

idênticos. Este resultado encontra-se de acordo com o discutido previamente na secção 3.5.2.1.

Em relação ao custo de *download*, trata-se de um evento único no início da vida do nó, um custo que vai sendo diluído no tempo à medida que o nó permanece mais tempo no sistema. Por esta razão verificamos o pico inicial e um decréscimo progressivo à medida que a sessão aumenta.

Na tabela 4.1 apresentamos os resultados obtidos nas diversas simulações.

Taxa de Chegada/Vizinhança	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$	Estimativa
1.6	50	49	51	50	35
3.2	97	99	-	-	70

Tabela 4.1: Largura de banda despendida em média no *download* e *upload* das bases de dados inicial de endereços (bytes/s)

Analisando os resultados verificamos que estes se mantém idênticos ao longo das diferentes vizinhanças o que seria de esperar uma vez que a visibilidade de endereços, chaves e chaves dos nós continua a ser completa e não sofre qualquer influência da vizinhança do nó.

Observando o valor estimado de acordo com o que foi discutido na secção 3.5.2.1 referente aos custos do descarregamento da base de dados inicial, verificamos que este valor fica um pouco abaixo dos resultados obtidos um facto que podemos atribuir à não contabilização do *overhead*. Como seria de esperar um aumento da taxa de participantes

traduz-se num aumento da largura de banda despendida. Adicionalmente verificamos que o comportamento modelado pela nossa estimativa é adequado uma vez que se verifica uma aumento na proporção esperada quando aumentamos a taxa de chegada para o dobro.

4.2.1.2 Base de Dados de Filtros

O descarregamento da base de dados de filtros é um dos aspectos que sofre influência directa da redução da visibilidade dos nós. Quanto menor a vizinhança dos nós, menores os custos associados ao *download* e *upload* da base de dados de filtros.

No gráfico 4.2 acima, podemos observar o comportamento da largura de banda despendida no *download* e *upload* da base de dados de filtros.

Verificamos em termos de *download* um comportamento em tudo semelhante ao que se verifica no *download* da base de dados de endereços - um *download* cujo peso é diluído à medida que o nó permanece no sistema. Em termos de *upload* verificamos no gráfico que os custos de *upload* da base de dados de filtros mantém-se constantes ao longo do tempo de sessão dos nós, o que está de acordo com a ausência do mecanismo de balanceamento descrito na secção do descarregamento da base de dados de endereços.

Na tabela 4.2 apresentamos os resultados obtidos nas várias simulações experimentais face à estimativa de acordo com o discutido previamente na secção 3.5.2.1.

Taxa de Chegada/Vizinhança	$\frac{1}{40}$ (estimativa)	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$
1.6	11 (10)	23 (20)	45 (40)	100 (80)
3.2	22 (22)	51 (40)	-	-

Tabela 4.2: Largura de banda despendida em média no *download* e *upload* das bases de dados inicial de filtros (bytes/s)

Como seria de esperar à medida que diminuímos o tamanho da vizinhança, os custos obtidos experimentalmente diminuem, validando assim a noção que motivou esta redução. Os valores obtidos apresentam um comportamento e magnitude de acordo com os valores estimados revelando que a expressão obtida durante a nossa análise é adequada.

4.2.1.3 Variância

Nas secções acima apresentamos os resultados em termos de valores médios, mas importa-nos saber também qual a variância deste resultados. No gráfico abaixo podemos verificar a variância associada aos custos do descarregamento das bases de dados iniciais. Verificamos que para estes descarregamentos a variância é realmente significativa nos custos de *upload*. Isto indica-nos que apesar dos esforços feitos para balancear a carga entre os participantes no que toca ao envio das bases de dados existe sempre a possibilidade de existirem nós que não realizam trabalho uma vez que a escolha é completamente aleatória e o sistema de uma dimensão considerável. Como seria de esperar não verificamos a

existência de uma variância tão significativa nos custos de *download* uma vez que todos os nós têm de realizar os descarregamentos referidos.

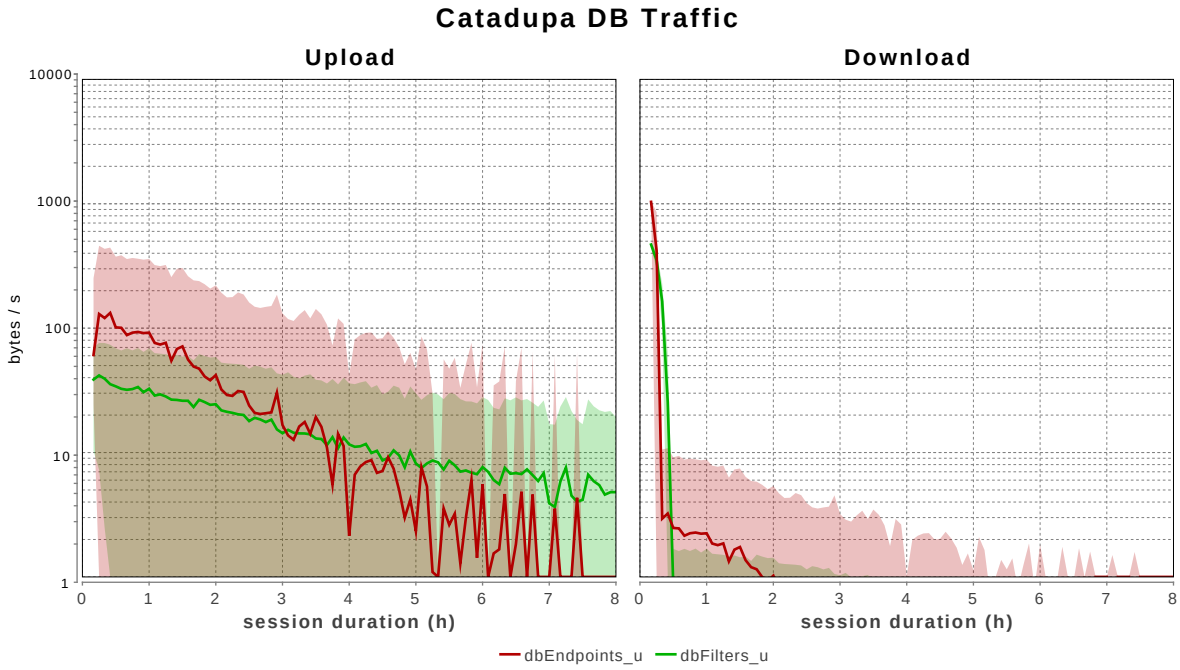

Figura 4.3: Variância do custo do descarregamento da base de dados inicial $r = 1.6$, $v = \frac{1}{20}$

4.2.2 Custo do novo Algoritmo de Filiação

O algoritmo de filiação, na medida em que é responsável pela disseminação das novas entradas e saídas do sistema, trata-se do aspecto do Livefeeds que em grande parte motivou esta dissertação. Nesta secção apresentamos os resultados experimentais obtidos referentes aos custos de largura de banda usada nas disseminações de novas entradas, saídas e filtros.

No gráfico 4.4 podemos verificar o padrão de comportamento da largura de banda despendida nos processos de filiação. Podemos verificar como seria de esperar que os custos relativos a este aspecto do sistema se mantém relativamente constantes ao longo do tempo de sessão. Este resultado tem como razão a distribuição relativamente constante das ocorrências das disseminações ao longo do tempo. Desta forma, um nó que está pouco tempo no sistema realiza tantas disseminações por unidade tempo quando um nó que permanece um maior período de tempo em sessão.

Nas seguintes secções apresentamos de forma tabelada os resultados referentes às várias simulações efectuadas bem como a comparação com os valores estimados.

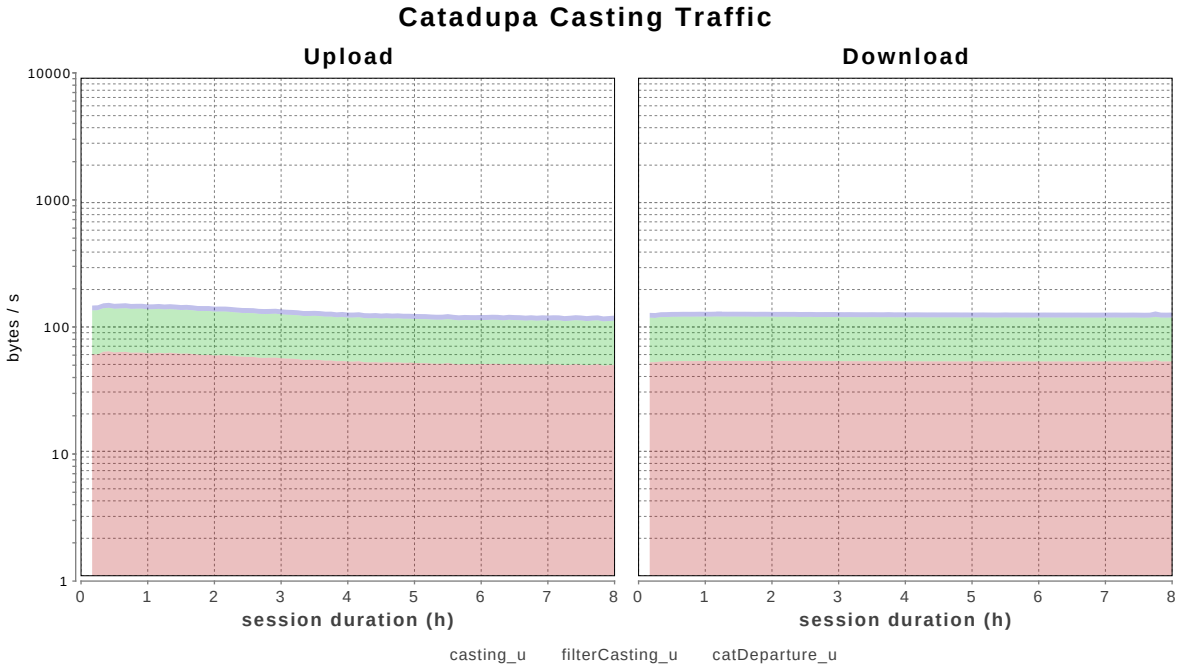

Figura 4.4: Custo do algoritmo de filiação $r = 1.6$, $v = \frac{1}{10}$

4.2.2.1 Disseminação de Novas Entradas

Na tabela 4.3 podemos verificar os custos obtidos experimentalmente para a disseminação de novas entradas.

Taxa de Chegada/Vizinhança	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$	Estimativa
1.6	52	51	51	52	42
3.2	100	100	-	-	74

Tabela 4.3: Largura de banda despendida em média (*download* e *upload*) nas disseminações de novas entradas (bytes/s)

A disseminação de novas entradas, à semelhança do que acontece com o descarregamento da base de dados inicial de endereços, não é influenciada pela vizinhança do nó uma vez que as disseminações continuam a ocorrer ao nível de todos os participantes. Comprovamos, como seria de esperar, que quanto maior a taxa de chegada de participantes maior o custo da disseminação de novas entradas. Verificamos que a estimativa é adequada no que toca à modelação do comportamento dos resultados esperados apesar de ficar um pouco aquém dos resultados obtidos devido à análise simplificada do *overhead* existente.

4.2.2.2 Disseminação de Filtros

Apresentamos na tabela 4.4 os resultados obtidos para o custo da disseminação de filtros.

Taxa de Chegada/Vizinhança	$\frac{1}{40}$ (estimativa)	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$
1.6	18 (15)	36 (30)	67 (59)	137 (118)
3.2	40 (29)	80 (59)	-	-

Tabela 4.4: Largura de banda despendida em média (*download* e *upload*) nas disseminações de filtros (bytes/s)

Observando a tabela concluímos que os valores estimados, de acordo com a secção 3.5.1, se aproximam dos valores obtidos experimentalmente. Como seria de esperar, quanto menor a vizinhança do nó, menores os custos associados a disseminação de filtros. Esta redução era o objectivo da redução de visibilidade dos nós consistindo num decréscimo das necessidades de largura de banda face à versão anterior.

4.2.2.3 Disseminação de Saídas

A disseminação de saídas encontra-se ligada à disseminação de novas entradas uma vez que são a mesma disseminação. Um evento de agregação de um sequenciador contém tanto a informação dos nós que entraram como dos nós que saíram. No entanto, contabilizamos separadamente o custo da disseminação de saídas para melhor avaliar a sua componente nos custos do sistema Livefeeds.

Na tabela 4.5 apresentamos os custos referentes à componente da disseminação de saídas.

Taxa de Chegada/Vizinhança	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$	Estimativa
1.6	11	10	10	10	10
3.2	20	20	-	-	19

Tabela 4.5: Largura de banda despendida em média (*download* e *upload*) nas disseminações de saídas (bytes/s)

Comparando as nossas estimativas, de acordo com a secção 3.5.3, verificamos que ficamos próximos dos valores obtidos experimentalmente. O valor mantém-se idêntico para diferentes valores de vizinhança uma vez que esta não influencia a disseminação de saídas dado que é uma disseminação que ocorre a nível de todos os participantes.

4.2.2.4 Variância

Em termos da variância dos custos relativos às disseminações do algoritmo de filiação podemos verificar que esta toma valores relativamente baixos. Em termos de *upload*, este valor é fruto dos mecanismos postos em prática com vista ao balanceamento de carga nas disseminações dos eventos de entradas/saídas, nomeadamente o deslocamento aleatório que existe associado ao intervalo global em cada disseminação de modo a que não sejam sistematicamente os mesmos nós a ser folhas da árvore de disseminação. Na disseminação de filtros, o facto desta ser feita num intervalo em função da chave do nó que a

inicia resulta também num balanceamento de carga natural. Em relação ao *download* a variância toma, como seria de esperar, um valor ainda mais reduzido uma vez que numa disseminação todos os nós tem que receber sempre uma mensagem.

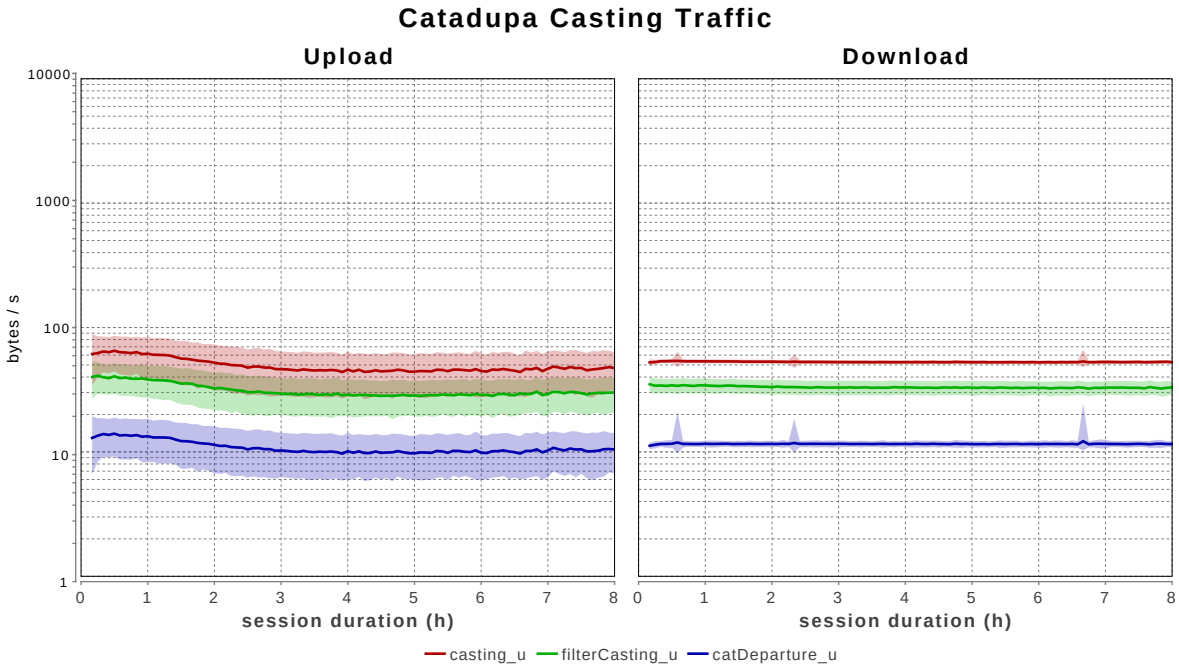

Figura 4.5: Variância do custo do algoritmo de filiação $r = 1.6$, $v = \frac{1}{20}$

4.2.3 Reparação Epidémica

A reparação epidémica é um aspecto bastante sensível do sistema Livefeeds. Os seus custos podem variar em grande medida consoante a susceptibilidade que o sistema tem para ocorrência de falhas. Na secção 3.5.4.1 referimos 30 segundos como um valor possível para o período de reparação. Na prática implementamos um período variável que se adapta em função dos resultados de cada reparação. Isto é, se o nó concluir que está a reparar um número muito reduzido de falhas pode então alargar o período de reparação para suavizar o impacto que este tem na largura de banda despendida. Devido à existência deste mecanismo e a dificuldade em estimar a probabilidade de falha no sistema é difícil de validar experimentalmente as estimativas propostas.

4.2.3.1 Reparação de Endereços

No gráfico 4.6 podemos observar o padrão de comportamento os consumos referentes às reparações epidémicas. Verificamos que estes são relativamente constantes ao longo do tempo de sessão do nó, um resultado que esperamos uma vez que o processo de reparação é periódico. Observamos inicialmente nos primeiros momentos de vida dum

nó, uma subida dos custos até à estabilização referida. Esta situação deve-se ao facto de que os nós só podem responder a uma reparação depois de obterem a base de dados de filtros. Caso contrário, estaríamos a permitir que a reparação funcionasse como *download* da base de dados inicial o que seria ineficiente e levaria um trabalho de *upload* excessivo para os nós que participariam nessas reparações.

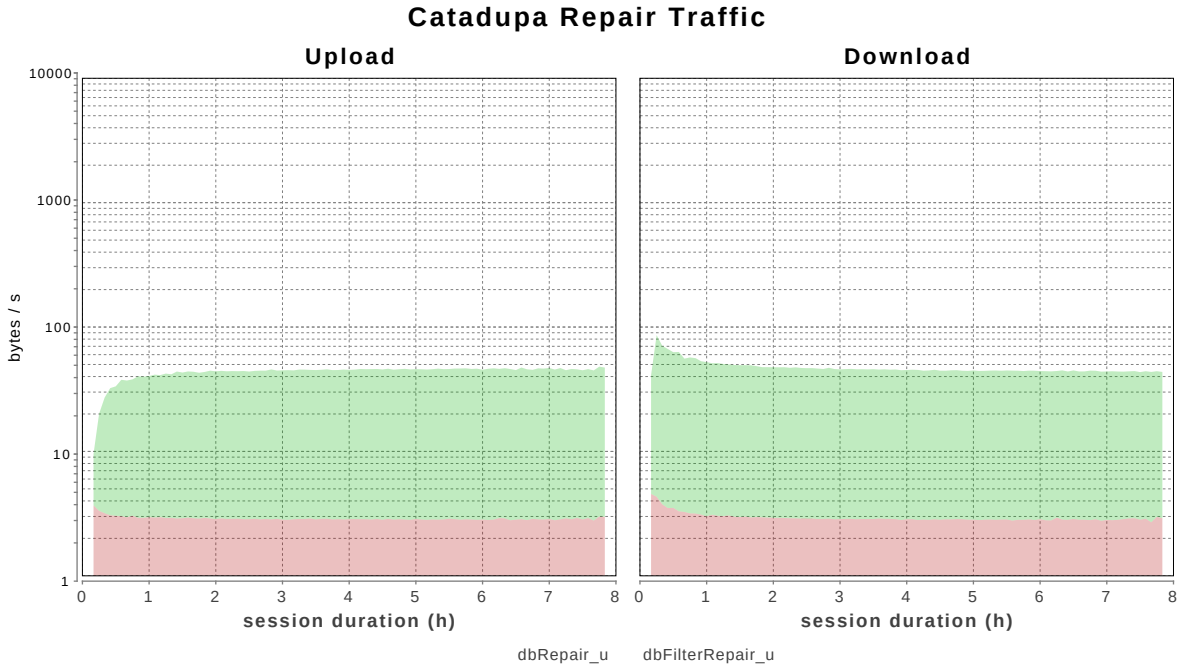

Figura 4.6: Custo da reparação epidémica $r = 1.6$, $v = \frac{1}{20}$

Na tabela 4.6 apresentamos os resultados obtidos referentes aos custos da reparação de endereços comparativamente à nossa estimativa ($p_{falha} = 0.05$).

Taxa de Chegada/Vizinhança	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$	Estimativa
1.6	2	2	2	2	16
3.2	20	20	-	-	21

Tabela 4.6: Largura de banda despendida em média (*download* e *upload*) na reparação de endereços (bytes/s)

Podemos verificar como seria de esperar que a reparação de endereços não depende da vizinhança do nó, uma vez que esta afecta apenas os filtros. Adicionalmente, a noção de árvores de agregação com vários níveis de sequenciadores reduz a ocorrência de omissões resultantes de disseminações concorrentes contribuindo para a redução do custo associado às reparações. A baixa probabilidade de falhas associadas à abordagem de árvores de agregação e o separar dos filtros desta reparação resultam num custo bastante reduzido. Observamos também que o aumento da taxa de participantes produz um efeito mínimo uma vez que a probabilidade de falhas é bastante baixa. Ainda assim,

caso esta aumente podemos esperar um aumento da influência da taxa de chegada de participantes.

4.2.3.2 Reparação de Filtros

Na tabela 4.8 apresentamos os resultados obtidos referentes aos custos da reparação de filtros comparativamente à nossa estimativa ($p_{falha} = 0.05$).

Taxa de Chegada/Vizinhança	$\frac{1}{40}$ (estimativa)	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$
1.6	26 (9)	43 (9)	73 (10)	125 (13)
3.2	42 (9)	75 (10)	-	-

Tabela 4.7: Largura de banda despendida em média (*download* e *upload*) na reparação de filtros (bytes/s)

O custo associado aos filtros passou então para a reparação de filtros que, como podemos observar, apresenta uma sensibilidade maior do que a estimada em relação à vizinhança do nó. Os valores obtidos apresentam uma diferença significativa em relação a nossa estimativa. Como foi discutido prever estes resultados é difícil uma vez que não sabemos à partida a probabilidade de ocorrerem falhas no sistema. Podemos no entanto verificar se a mesma modela o comportamento dos custos de forma correcta. Considerando o valor obtido para os custos associados à reparação de filtros na vizinhança $\frac{1}{40}$, determinamos a partir deste o número médio de filtros em falta, de acordo com a análise da secção 3.5.4.2.

$$\overline{b_u} = \overline{b_d} = \frac{n_{f_{falta}} \cdot (m_f + m_{fk}) + 3m_h}{T_{rep_k}} \ (bytes/s) \tag{4.2}$$

$$n_{f_{falta}} = \frac{\overline{b_u} \cdot T_{rep_f} - 3m_h}{(m_f + m_{fk})} \approx 1 \tag{4.3}$$

Este valor implica uma probabilidade superior a 1 mas temos que atender ao facto de que na prática nem todas as reparações tem sucesso e as falhas podem-se acumular entre reparações.

Com base no valor deduzido para o número médio de filtros em falta na simulação de vizinhança $\frac{1}{40}$ e considerando a probabilidade de falha constante entre simulações podemos actualizar as estimativas para as restantes vizinhanças estimando $n_{f_{falta}}$ como,

$$n_{f_{falta}} = \frac{40 \times 1 \cdot v \cdot r}{1.6} \tag{4.4}$$

Verificamos deste modo que a expressão obtida modela correctamente o comportamento dos custos da reparação de filtros em função da taxa de chegada de participantes e da vizinhança do nó. No entanto, a dificuldade referida em determinar à partida o número médio de filtros em falta impossibilita a estimativa de valores sem recorrer ao uso

Taxa de Chegada/Vizinhança	$\frac{1}{40}$	$\frac{1}{20}(estimativa)$	$\frac{1}{10}$	$\frac{1}{5}$
1.6	26	43(43)	73(79)	125(150)
3.2	42(43)	75(79)	-	-

Tabela 4.8: Largura de banda despendida em média (*download* e *upload*) na reparação de filtros comparativamente às estimativas actualizadas (bytes/s)

de observações experimentais.

Embora exista uma redução dos custos em função da vizinhança, esta redução é limitada pela complexidade da representação do conhecimento de filtros que é mais pesada do que quando dos filtros faziam parte dos eventos de agregação globais. O *overhead* diminui também esta redução indicando mais uma vez a importância de definir correctamente o período da reparação.

4.2.3.3 Variância

Em termos de variância nos custos das reparações, tal como foi discutido no capítulo da análise, esperamos que um nó participe numa reparação aproximadamente tantas vezes como iniciador quanto receptor uma vez que este é um processo periódico efectuado por todos os nós em que estes escolhem um outro participante de forma aleatória. Esta expectativa encontra-se de acordo com os resultados que podemos observar no gráfico 4.7.

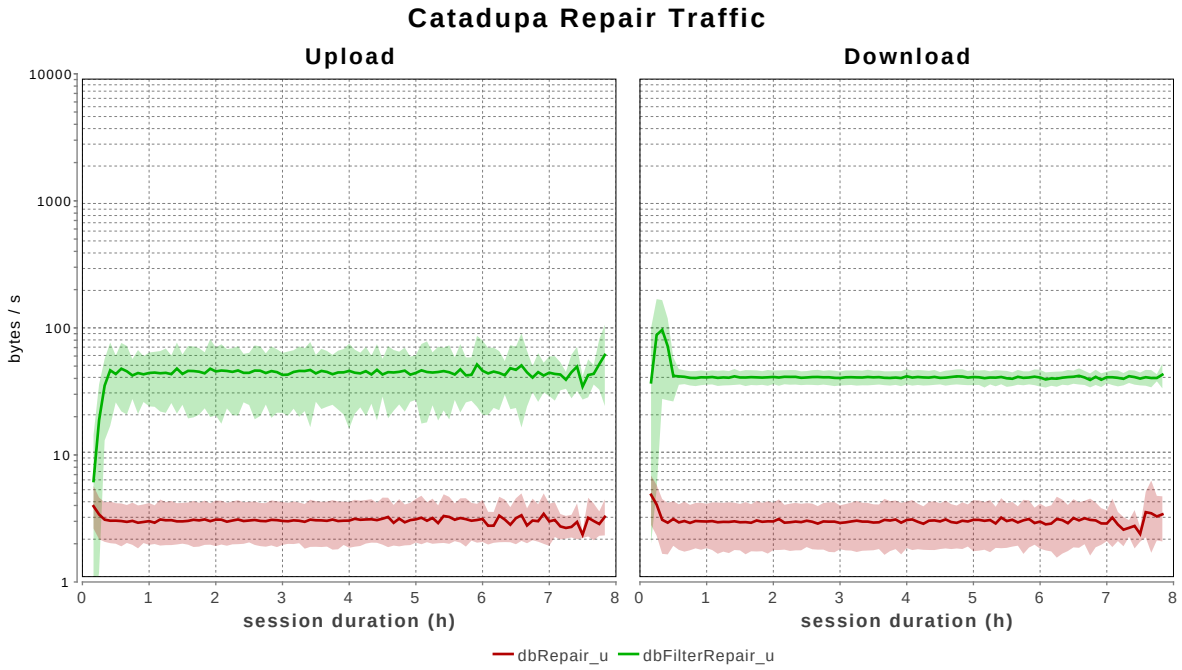

Figura 4.7: Variância do custo da Reparação Epidémica $r = 1.6$, $v = \frac{1}{20}$

4.2.4 Custo do Algoritmo de Disseminação Filtrada

O algoritmo de disseminação filtrada, alterado nesta nova versão devido à introdução da noção de vizinhança associada à redução de visibilidade, foi analisado na secção 3.5.5. Na referida secção concluímos que o algoritmo de disseminação filtrada não dependia da vizinhança do nó mas sim do número de participantes em cada disseminação. O número de participantes em cada disseminação depende da taxa de chegada de nós ao sistema e da popularidade dos eventos. Esta popularidade tem o valor, nas nossas simulações, de aproximadamente 0.5 como podemos comprovar no gráfico 4.8 que demonstra a distribuição da popularidade dos eventos ocorridos durante uma simulação.

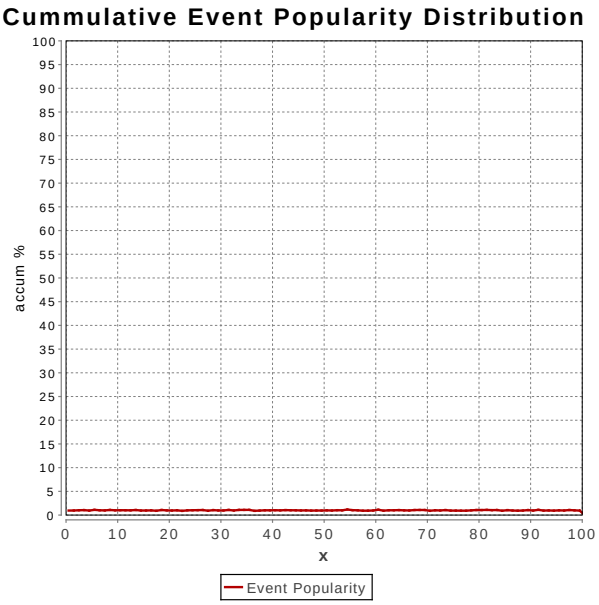

Figura 4.8: Distribuição da popularidade dos eventos ocorridos durante uma simulação

No gráfico 4.9 apresentamos o padrão de resultados verificado ao longo das várias simulações. Verificamos que a largura de banda despendida nas disseminações filtradas aumenta nos primeiros momentos de vida de um nó estabilizando posteriormente ao longo da sessão dos nós. Este facto deve-se a que o nó só pode participar numa disseminação filtrada quando completado o *download* da base de dados inicial de filtros. Como discutido já anteriormente, o *download* e *upload* associados a uma disseminação são semelhantes já que sendo o intervalo da disseminação alvo de um deslocamento aleatório a posição que o nó toma na árvore de disseminação encontra-se bem distribuída ao longo das várias disseminações que ocorrem.

Na tabela 4.9, apresentamos os resultados obtidos para as diferentes simulações bem como as estimativas de acordo com a análise da solução feita previamente.

De acordo com as conclusões na secção 3.5.6.3, o valor de falsos positivos era percentualmente baixo excepto para eventos de popularidade muito baixa isto leva-nos a concluir que apesar o número total de falsos positivos gerados variar consoante a vizinhança

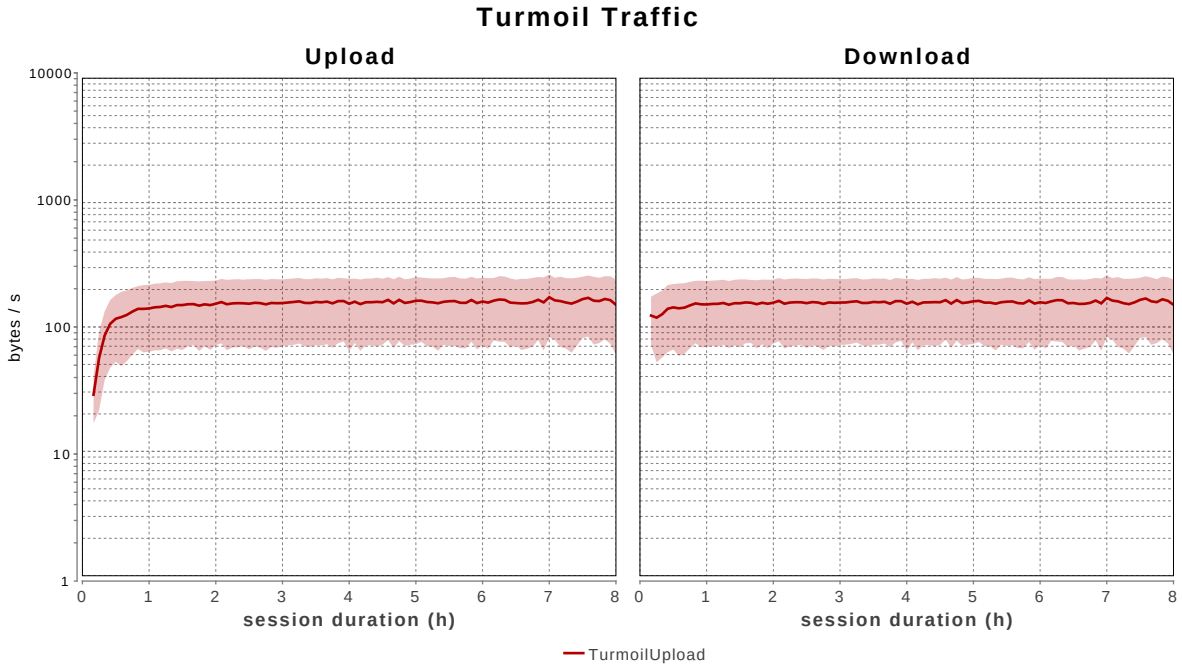

Figura 4.9: Custo do algoritmo de disseminação filtrada $r = 1.6$, $v = \frac{1}{10}$

Taxa de Chegada/Vizinhança	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$	Estimativa
1.6	175	170	162	170	111
3.2	181	175	-	-	113

Tabela 4.9: Largura de banda despendida em média (*download* e *upload*) nas disseminações de saídas (bytes/s)

do nó, o seu número é tão reduzido comparativamente ao número de participantes em cada disseminação para um valor de popularidade média perto de 50% que não se verificam alterações relevantes na largura de banda despendida em função da vizinhança dos nós.

Comprovamos nos valores tabelados o comportamento esperado de acordo com as nossas estimativas - o aumento da taxa de chegada de participantes tem um efeito mínimo nos custos associados ao algoritmo de disseminação filtrada. Desta forma, este depende principalmente da taxa de ocorrência de eventos no sistema e da popularidade dos mesmos.

Comparando os valores obtidos experimentalmente com a nossa estimativa verificamos que estes ficam abaixo dos resultados obtidos. Este facto pode ser atribuído ao facto de que com o intuito de evitar falsos negativos, os nós podem rejeitar disseminar um evento caso a sua visão do sistema seja uma restrição da visão do nó do qual receberam o evento. Isto implica que a árvore não é totalmente regular e que os custos aumentam comparativamente aos valores estimados.

4.2.5 Introdução de Falsos Positivos

Como discutido anteriormente, ao reduzirmos a visibilidade dos participantes estamos a criar uma situação em que existe a possibilidade de ocorrerem falsos positivos. Nesta secção apresentamos os resultados obtidos no que diz respeito à introdução de falsos positivos no algoritmo de disseminação filtrada.

4.2.6 Falsos Positivos e Envios Cegos

Nos gráficos 4.10 a 4.13 mostramos os resultados obtidos experimentalmente em termos de falsos positivos gerados em média por cada nível das árvores de disseminação. Comparando as curvas obtidas experimentalmente com as curvas teóricas previamente apresentadas (Figura 3.6 da secção 3.5.6.1), verificamos uma aproximação razoável, significando que as estimativas feitas adequam-se aos resultados experimentais. Comprovamos a influência directa da vinhança do nó na geração de falsos positivos e o seu comportamento desde o momento inicial em que temos uma subida do número total de falsos positivos por nível, uma vez que mais nós realizam envios a cada nível, seguido de um decréscimo à medida que os intervalos ficam cada vez mais pequenos até ficarem totalmente contidos nos intervalos de vizinhança dos nó.

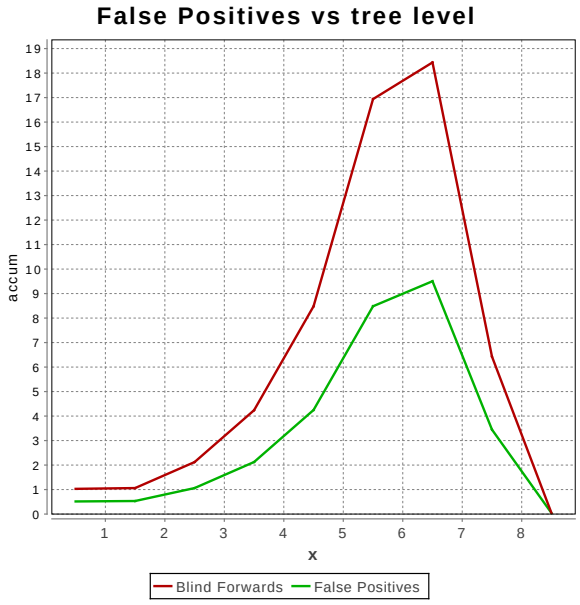

Figura 4.10: Falsos positivos introduzidos por nível da árvore de disseminação $v = \frac{1}{40}$

Importante a observar nos gráficos apresentados é a diferença entre o que consideramos um falso positivo em teoria e um falso positivo em termos práticos. Em teoria, qualquer envio feito sem conhecimento do filtro dum nó resulta num potencial falso positivo. Em termos práticos, referindo essa situação como o envio cego, contabilizamos quantos desses envios resultam realmente num falso positivo. Isto é, em quantos desses o nó receptor não aceitaria o evento. Este valor depende da popularidade do evento e

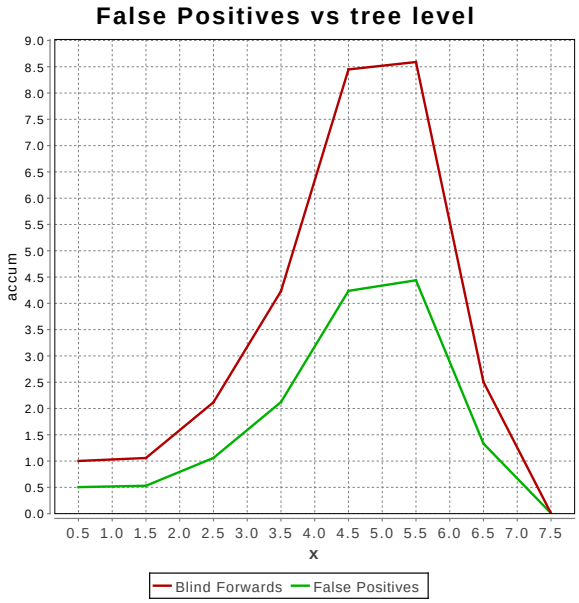

Figura 4.11: Falsos positivos introduzidos por nível da árvore de disseminação $v = \frac{1}{20}$

verifica-se graficamente um decréscimo do número de falsos positivos a cada nível em aproximadamente 50 %, aquela que é a popularidade média dos eventos que ocorrem no sistema.

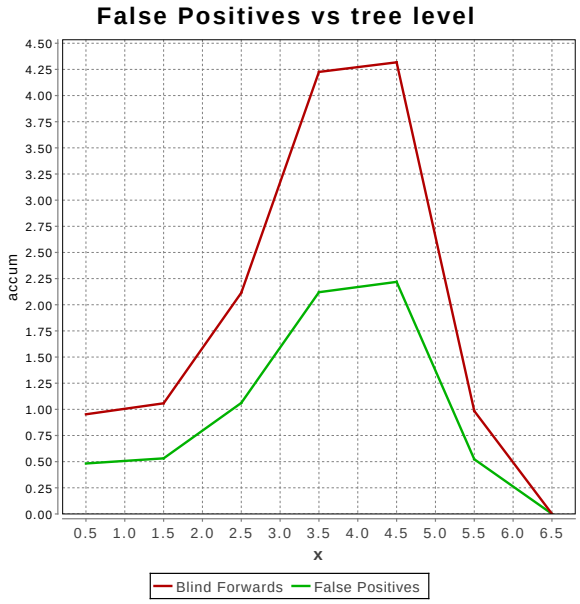

Figura 4.12: Falsos positivos introduzidos por nível da árvore de disseminação $v = \frac{1}{10}$

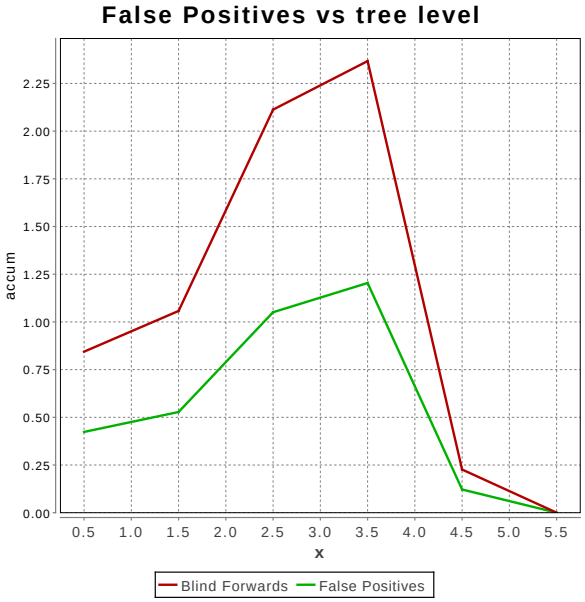

Figura 4.13: Falsos positivos introduzidos por nível da árvore de disseminação $v = \frac{1}{5}$

4.2.7 Percentagem de Falsos Positivos face ao número total de envios

Por sua vez, em termos de percentagem de falsos positivos em relação ao número total de envios associados a uma disseminação, comprovamos que este valor depende da popularidade de um evento. Nos gráficos 4.14 a 4.17 verificamos que esta só tem valores mais significativos para eventos de reduzida popularidade. Este facto, está de acordo com discutido anteriormente na secção 3.5.6.3. Podemos ainda nestes gráficos verificar a redução dessa percentagem à medida que aumenta a vizinhança do nó.

Comprovamos assim a existência de dois factores associados à popularidade que contribuem para a suavização do impacto percentual da ocorrência de falsos positivos num sistema. Quanto mais elevada a popularidade média dos eventos num sistema maior serão as árvores de disseminação diminuindo a percentagem de falsos positivos uma vez que a partir de certo nível não são gerados falsos positivos e, quanto maior a popularidade, também mais elevada será a probabilidade de o envio cego não resultar num falso positivo em termos práticos.

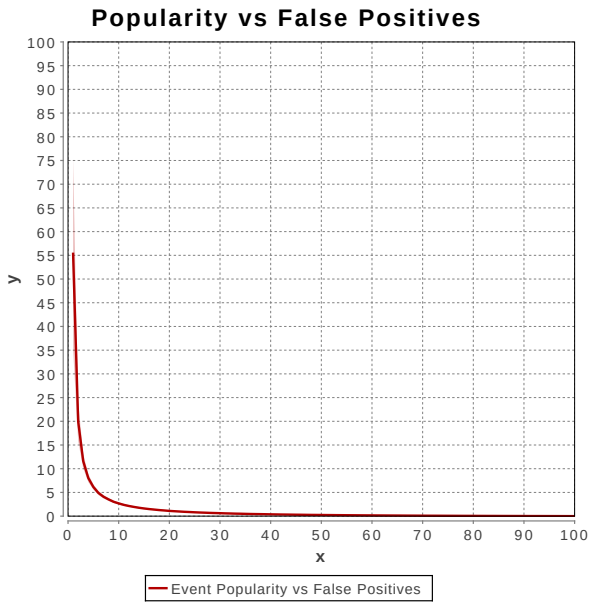

Figura 4.14: Percentagem de falsos positivos em função da popularidade do evento $v = \frac{1}{40}$

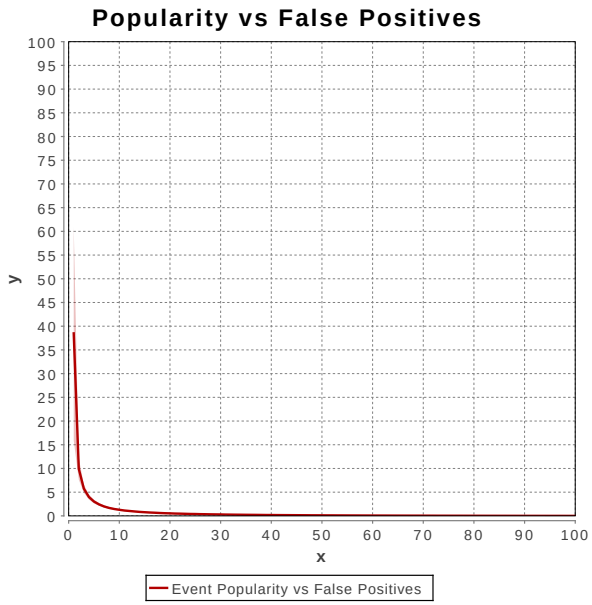

Figura 4.15: Percentagem de falsos positivos em função da popularidade do evento $v = \frac{1}{20}$

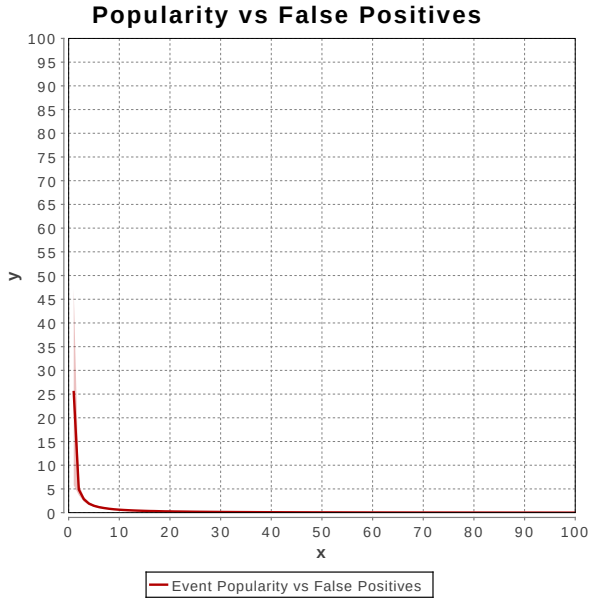

Figura 4.16: Percentagem de falsos positivos em função da popularidade do evento $v = \frac{1}{10}$

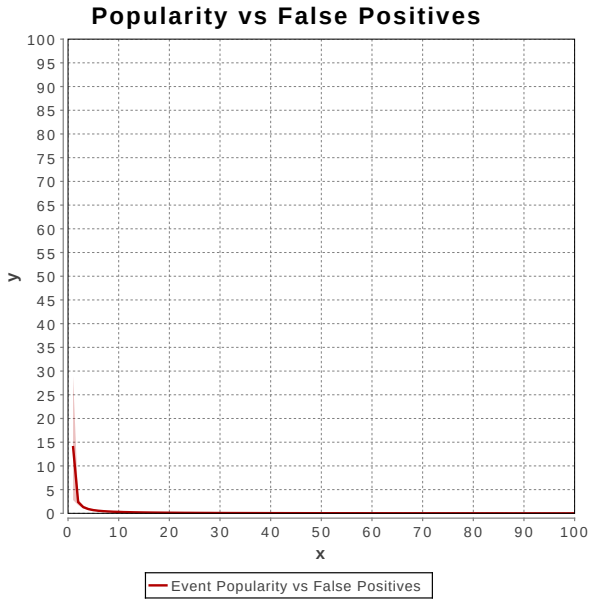

Figura 4.17: Percentagem de falsos positivos em função da popularidade do evento $v = \frac{1}{5}$

4.2.8 Comparação com a versão de Visibilidade Completa

Nas secções acima apresentámos os resultados obtidos experimentalmente. Estes demonstram a validade da solução proposta e confirmam as nossas hipóteses de que uma redução da visibilidade dos nós terá efeitos positivos no custo associado a todo o processo de filiação. Resta-nos agora comparar os resultados obtidos com resultados referentes à versão de visibilidade completa.

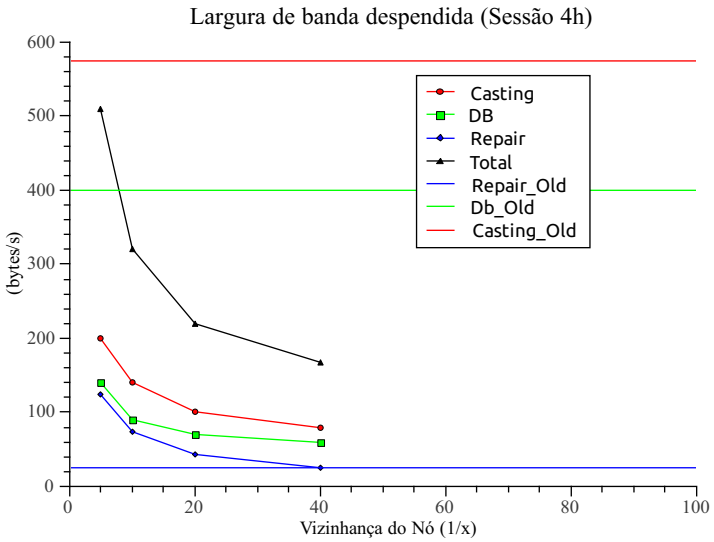

Figura 4.18: Comparação entre os resultados obtidos para a versão de visibilidade parcial e a versão de visibilidade completa

No gráfico 4.18 comparamos os resultados obtidos nesta dissertação para diferentes valores de vizinhança com resultados conhecidos da versão de visibilidade completa para uma mesma taxa de participantes. Verificamos que ocorreu um decréscimo considerável em todos os aspectos da filiação com excepção à reparação. Esta excepção revela-nos o impacto da separação do processo de reparação dos filtros dos restantes objectos. Como discutido anteriormente, a dificuldade em representar de forma compacta a história de eventos de disseminação de filtros leva a que o *overhead* associado a uma reparação de filtros seja superior e tenha um impacto significativo na largura de banda despendida.

Os resultados obtidos e apresentados neste capítulo demonstram a validade e viabilidade da solução proposta. A redução da visibilidade produz uma redução significativa do custo total do algoritmo de filiação confirmando as nossas expectativas. Por sua vez, a introdução de falsos positivos revela-se relativamente reduzida quando consideramos sistemas com um número relevante de participantes. Este facto, consiste num ponto encorajador para esta abordagem uma vez que esta acomoda um maior dinamismo em termos de entradas que, resultando em sistemas de maior escala dilui progressivamente o impacto dos falsos positivos no sistema. Ainda assim, a ausência de falsos positivos é

impossível de garantir e os mecanismos discutidos para a sua mitigação são probabilísticos. Deste modo, no caso de ser necessário garantir por completo a sua ausência a solução não seria válida. No entanto este é um caso extremo. Para a maior parte dos casos a introdução de falsos positivos à escala aqui demonstrada é tolerável e não desqualifica a solução apresentada de ser potencialmente um melhoramento do sistema Livefeeds.

5

Conclusões

Motivada pela crescente relevância de sistemas de disponibilização de conteúdos face ao aumento do volume de informação disponível na *web*, esta dissertação surgiu no âmbito do sistema Livefeeds. Este sistema, consistindo num sistema *peer-to-peer* com vista ao suporte de aplicações que seguem o paradigma editor/assinante, oferece uma alternativa às soluções mais dispendiosas e centralizadas baseadas em servidores.

Seguindo os princípios das redes sobrepostas estruturas (redes lógicas bem organizadas implementadas sobre a infraestrutura real), o sistema Livefeeds permite aos utilizadores definirem o seu interesse em eventos por meio de filtros. Posteriormente, quando ocorre um evento, este é disseminado pelos participantes através duma árvore formada na altura pelos participantes cujos filtros aceitam o evento, naquele que apresentámos como o problema da disseminação filtrada.

A organização do sistema baseia-se num algoritmo de filiação que lida com a entrada de cada participante no sistema. Este visava produzir em todos os participantes uma visão completa dos restantes nós presentes no sistema. Desta forma os nós poderiam facilmente testar os filtros dos restantes participantes durante a disseminação filtrada e escolher apenas os nós cujo filtro aceita o evento.

O algoritmo de filiação de visibilidade completa dissemina as novas entradas por todos os participantes do sistema. Caso a taxa de participantes seja relativamente alta o seu custo pode tornar-se elevado. Com a motivação de reduzir este custo e mediante a ideia de que os filtros são os objectos de maior dimensão associados a um participante, propusemos um algoritmo de filiação com visibilidade parcial. Nesta solução, por oposição à versão anterior, os nós conhecem apenas uma parte dos filtros presentes no sistema. Pretendíamos assim reduzir os custos de largura de banda associados ao algoritmo de filiação. Ainda assim, existiria um compromisso - a possibilidade de entregar eventos

a nós cujo filtro desconhecemos. Esta situação vista como indesejada na prática, pois queremos que os nós apenas realizem trabalho em eventos que realmente desejam.

Apresentámos nesta dissertação, uma descrição pormenorizada da solução de visibilidade parcial. A solução completa foi alvo duma análise através da qual obtivemos métricas importantes que permitem modelar o padrão de comportamento das várias componentes do sistema. A mesma análise justifica parametrizações que servem de ponto de partida à implementação de diferentes configurações do Livefeeds permitindo identificar à partida as necessidades aproximadas em termos de largura de banda do sistema nos diferentes aspectos dos algoritmos de filiação, reparação e disseminação filtrada, bem como o impacto em termos de falsos positivos.

A solução foi posteriormente materializada num protótipo simulado através do qual verificámos a viabilidade desta solução e comprovámos os benefícios ao nível de custos de largura de banda previstos e que foram a motivação desta dissertação. Verificámos também a existência de falsos positivos, no entanto, tanto na nossa análise teórica como nos resultados obtidos experimentalmente, verificamos que a sua ocorrência é reduzida em termos percentuais para uma taxa de chegada de participantes relevante. A ocorrência de falsos positivos é ainda mitigada pela popularidade média dos eventos que ocorre num sistema e pela eventual redundância de filtros existentes no mesmo. Estes factos levam-nos a concluir que a solução apresentada é bastante relevante para a evolução do sistema livefeeds na direcção de suportar um maior dinamismo.

5.1 Contribuições

Como resultado desta dissertação foram apresentadas as seguintes contribuições:

- Desenho de um algoritmo de filiação com visibilidade parcial
- A adaptação do algoritmo de disseminação filtrada
- Uma análise detalhada da solução proposta
- A implementação de um protótipo simulado
- Validação experimental da solução através do referido protótipo

5.2 Trabalho Futuro

A versão do Livefeeds desenhada e estudada nesta dissertação apresenta um benefício claro de redução de consumos de largura de banda. Esta redução é resultado da redução da visibilidade dos nós em relação aos filtros dos restantes participantes. Esta dissertação abre as portas a uma corrente de investigação no que diz respeito à aplicação da visibilidade parcial ao sistema Livefeeds. Com base nisto, alguns pontos interessantes a realizar no futuro seriam:

- Redução da visibilidade de endereços
- Análise mais detalhada do impacto da redundância ao nível da disseminação filtrada e quanto esta pode ser realmente usada para mitigar a ocorrência de falsos positivos
- Teste da solução em ambiente real

Bibliografia

[1] R. Baldoni, C. Marchetti, A. Virgillito, and R. Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 437–446. IEEE, 2005.

[2] S. Birrer and F.E. Bustamante. Resilient peer-to-peer multicast without the cost. In *Proc. SPIE*, volume 5680, pages 113–120. Citeseer, 2005.

[3] A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. Design and evaluation of a wide-area event notification service. 2003.

[4] M. Castro, M. Costa, and A. Rowstron. Peer-to-peer overlays: structured, unstructured, or both? 2004.

[5] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 85–98. USENIX Association, 2005.

[6] M. Castro, P. Druschel, A.M. Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. *Group Communications and Charges*, pages 47–57, 2003.

[7] M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[8] R. Chand and P. Felber. XNET: a reliable content-based publish/subscribe system. In *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*, pages 264–273. IEEE, 2004.

[9] R. Chand and P.A. Felber. Efficient subscription management in content-based networks. In *Third International Workshop on Distributed Event-Based Systems*, page 14. Citeseer, 2004.

[10] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[11] F. Dabek, E. Brunskill, M. F. Kaashoek, and D. Karger. Building peer-to-peer systems with chord, a distributed lookup service. *ACM SIGCOMM*, August 2001.

[12] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[13] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. HotOS VIII*, pages 75–80. Citeseer, 2001.

[14] P. Goering and G. Heijenk. Service discovery using Bloom filters. In *Proc. Twelfth Annual Conference of the Advanced School for Computing and Imaging, Belgium*. Citeseer, 2006.

[15] S. Guha, N. Daswani, and R. Jain. An experimental study of the skype peer-to-peer voip system. In *Proc. of IPTPS*, volume 6. Citeseer, 2006.

[16] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation-Volume 1*, page 9. USENIX Association, 2004.

[17] Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues. One hop lookups for peer-to-peer overlays. *MIT Laboratory for Computer Science*, 2004.

[18] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. *IPTPS*, 2003.

[19] M. Kaashoek and D. Karger. Koorde: A simple degree-optimal distributed hash table. *Peer-to-Peer Systems II*, pages 98–107, 2003.

[20] J. Legatheaux and M.S. Duarte. Routing Algorithms for Content-based Publish/-Subscribe Systems. 2007.

[21] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM, 2002.

[22] Simão Mata, J. Legatheaux Martins, Sérgio Duarte, and Margarida Mamede. Análise do custo e da viabilidade de um sistema p2p com visibilidade completa.

[23] P.R. Pietzuch and J.M. Bacon. Hermes: A distributed event-based middleware architecture. 2002.

[24] C.G. Plaxton, R. Rajaraman, and A.W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280, 1999.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *SIGCOMM*, August 2001.

[26] M. Ripeanu and I. Foster. Peer-to-peer architecture case study: Gnutella network. *University of Chicago, Chicago*, 2001.

[27] R. Rodrigues and C. Blake. When multi-hop peer-to-peer lookup matters. *Peer-to-Peer Systems III*, pages 112–122, 2005.

[28] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. IFIP/ACM Middleware*, November 2001.

[29] A. Rowstron, A.M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. *Networked Group Communication*, pages 30–43, 2001.

[30] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202. ACM, 2006.

[31] D. Stutzbach, R. Rejaie, and S. Sen. Characterizing unstructured overlay topologies in modern P2P file-sharing systems. *Networking, IEEE/ACM Transactions on*, 16(2):267–280, 2008.

[32] Sasu Tarkoma. *Overlay Networks, Towards Information Networking*. CRC Press, 2010.

[33] P. Triantafillou and I. Aekaterinidis. Content-based publish-subscribe over structured P2P networks. In *Proceedings of the 4th International Workshop on Distributed Event-Based Systems*. Citeseer, 2004.

[34] Y.M. Wang, L. Qiu, D. Achlioptas, G. Das, P. Larson, and H.J. Wang. Subscription partitioning and routing in content-based publish/subscribe systems. In *16th International Symposium on DIStributed Computing*, 2002.

[35] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.